AFRL-IF-RS-TR-2006-153
Final Technical Report
May 2006

SOFTWARE INFRASTRUCTURE TO SUPPORT DSAP (DYNAMIC SITUATIONAL AWARENESS AND PREDICTION) CAPABILITIES

RAM Laboratories, Inc.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2006-153 has been reviewed and is approved for publication.

APPROVED: /s/

DAWN A. TREVISANI
Project Engineer

FOR THE DIRECTOR: /s/

JAMES W. CUSACK
Chief, Information Systems Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	MAY 2006	Final Jun 2003 – Dec 2005

4. TITLE AND SUBTITLE
SOFTWARE INFRASTRUCTURE TO SUPPORT DSAP (DYNAMIC SITUATIONAL AWARENESS AND PREDICTION) CAPABILITIES

5. FUNDING NUMBERS
C - F30602-03-C-0112
PE - 65702F
PR - 459S
TA - MA
WU - 03

6. AUTHOR(S)
Robert McGraw

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
RAM Laboratories, Inc.
10525 Vista Sorrento Parkway, Suite 220
San Diego California 92121

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/IFSB
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2006-153

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Dawn A. Trevisani/IFSB/(315) 330-3657 Dawn.Trevisani @rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***
Today's C4I systems will be required to support faster-than-real-time predictive simulation that can determine possible outcomes by re-calibrating with real-time sensor data or extracted knowledge in real-time. This capability is known as a Dynamic Situation Assessment and Predication (DSAP) capability. The work under this contract involved developing a software infrastructure to support the implementation of a DSAP capability for decision aids. Specifically, this effort focused on developing enhancements for existing and evolving software frameworks by supporting capabilities that allow objects to be dynamically created, deleted and reconfigured, that allow simulations to be calibrated with live data feeds to provide state estimation, and that allow simulations to reduce overheads while supporting these mechanisms in order to continue to run in real-time. This effort examined the software framework used in support of the Joint Semi-Automated Forces (JSAF) program as a testbed for promoting this DSAP capability, implement these infrastructure improvements and integrate the technology with C4I systems.

14. SUBJECT TERMS
Dynamic Situation Assessment and Predication, DSAP, software infrastructure, predictive simulation, operationally focused simulation, C2 Decision Support, software frameworks, decision aid technologies.

15. NUMBER OF PAGES
50

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

List of Tables

1.0 Introduction

Military commanders desire timely battlefield information to make decisions based on the effects of their plans and the current operational picture. Often times the existing plan requires modification “on the fly” due to information that emerges through sensor detection and intelligence inputs as circumstances evolve in the operational environment. This “emerging” information causes decisions to be altered, assets to be re-tasked, and/or new alternatives to be considered. As such, tools and techniques are needed to better support (1) analysis in the context of this information, (2) training of Commanders and their staff in utilizing this information, and (3) use of this information to enhance operations. This Final Report details efforts of RAM Laboratories in developing a DSAP (Dynamic Situation Assessment and Prediction) software infrastructure that supports each of these goals.

1.1 Significance to the Air Force

The concept of DSAP grew out of John R. Surdu’s Simulation in Operations research project and prototype system (OpSim) by which he introduces the concept of operationally-focused simulation. Through this concept, he defends his notion that simulation used in real-time operational environments can be effective in supporting decision-makers. By embedding simulation within an operational setting, decision-makers can use simulation to plan operations, monitor current operations, determine deviations from a plan, predict outcomes, and project different outcomes.

DSAP is achieved by implementing two distinct functions: (1) Dynamic Situation Assessment, and (2) Prediction. Dynamic Situation Assessment is realized when methods and technologies are implemented to fuse simulation with emerging, real-world data in order to provide a current operational picture of the battlespace. For example, real-world data can be pulled from real-time Command Control Computers Communications Intelligence (C4I) databases and used to calibrate the estimated state of simulations reflecting the current state of operations. Prediction techniques can be employed by simulating alternate plans forward in time from the current state, providing predictive analysis of real-time battlefield effects to decision aids operating in real-time. Thus, the prediction capability improves the real-time planning process by providing a faster-than-real-time predictive assessment of Courses Of Action (COAs), alternate COAs, and operational effects on-the-fly to support "what if" scenarios.

While this DSAP capability does not exist, it can be realized by augmenting existing decision aids with faster-than-real-time simulation and advanced information technology concepts. To enable these capabilities, the software frameworks supporting these decision aid technologies must evolve. Specifically, enhancements are required for both new and legacy software frameworks to support functionality that allows objects to be dynamically created, deleted and reconfigured; allows simulations to be calibrated with live data feeds, and allows simulations to reduce overheads while supporting these mechanisms in order to continue to run in real-time. Section 1.1.1 discusses existing software infrastructure technologies and advanced information management functionality that can support DSAP.

1.1.1 Infrastructure Enhancements and Functionality Required for DSAP

This subsection details some of the infrastructure enhancements and functionality for software frameworks and infrastructure that can support DSAP functionality. A summary of some of this

functionality is presented in Table 2 and a discussion of each type of functionality is provided in subsequent sections.

Table 2: Relating DSAP Capabilities to Software Infrastructure Technologies.

SOFTWARE INFRASTRUCTURE TECHNOLOGY	CAPABILITY
Persistence	Provide the capability to save the simulation state at various points in a simulation execution.
Checkpoint/Restart	Provide the capability to Restart the Simulation from a Saved State.
Dynamic Object Creation/Deletion	Enter or remove new assets into a simulation and recalculate during run-time.
Live data feed Integration and Recalibration	Enter or remove new sensor or parametric information during run-time. Reconfigure scenario or vignette.
Rollback Framework	Rewind a simulation to a key point in time where new assets are discovered or re-tasked.
Rollforward/Lazy Cancellation	Fast-forward a simulation if new assets or re-tasking do not affect other assets.
Event Reparation	Repair events that are not affected by newly discovered or removed assets.
Multiple Replications	Utilizes distributed processing power to execute simulation replications faster.

1.1.1.1 Persistence and Checkpoint/Restart

Persistence and Checkpoint/Restart capabilities are important for enabling DSAP. The persistence capability allows a simulation to be "rolled back" to a specific checkpoint. At that checkpoint, assets playing in the simulation can be re-tasked to evaluate a new course of action. Persistence provides a simulation with the capability to save its state (as a checkpoint), and restart the simulation from any saved checkpoint. Persistence keeps track of memory allocations and pointer references in an internal database by allowing an object, and all of the objects that it references, to be packed into a buffer. During a restart, the buffer can be used to reconstruct the object. Even though the newly reconstructed objects may be instantiated in different memory locations, the persistence framework updates all affected pointer references to the new memory locations.

1.1.1.2 Dynamic Object Creation/Deletion

One of the key technologies that must be supported to enable DSAP is a dynamic object creation/deletion capability. The dynamic object creation/deletion capability allows simulation objects to be created or deleted dynamically during run-time. DSAP is supported by enabling assets to be dynamically created or deleted via user or machine interaction, thus allowing simulations to introduce new targets of opportunity and re-task assets "on-the-fly."

1.1.1.3 Rollback/Rollforward/Lazy Cancellation

DSAP requires a rollback framework that supports the rolling back of events to a given point in time. By rolling back time, simulations can reprocess information from a point where new direction can be taken. A DSAP capability also requires lazy cancellation and roll forward functionality. A touch/depend system can be used to implement lazy cancellation by automatically determining the state variables of a simulation that each event modifies and is dependent upon. Events will then rollforward when rollback-causing stragglers do not affect the outcome of processed events. This means that when the simulation state is rolled back (for the purpose of object creation/deletion or re-tasking), the rolled back state is saved and then rolled forward if those events are not effected by the new objects or re-tasking.

1.1.1.4 Event Reparation

DSAP implementations require techniques that support event reparation. When rolling a simulation back to address re-tasking issues or modify assets, events will be rolled back to a specific point in the simulation. Some of these rollbacks will be due to straggler events. Some events that are rolled back due to stragglers can be repaired. Implementing this technique for those events would reduce the number of cascading rollbacks by fixing the events instead of reprocessing them from scratch. This process will greatly improve the performance of many of the built-in events that are currently used for distributing data. By employing this strategy, performance improvements can allow simulations to execute at faster-than-real-time in order to provide real-time predictive inputs to decision aids.

1.1.1.5 Multiple Replications

DSAP may require methods for supporting the use of multiple replicated trials in cases where Monte Carlo or deterministic simulation is used. Monte Carlo simulations use pseudo random number generators to execute a simulated scenario many times. Statistical analysis is then performed on the collective outcomes to determine results. It is easy to achieve parallelism when executing Monte Carlo simulations by farming the replications to available processors.

1.1.1.6 Calibration with Live Data Feeds

DSAP requires a software infrastructure that can be integrated with and calibrated by live data feeds. The subsequent system maintains the capability to be updated by real-time data and automatically re-simulate scenarios based on emerging situational assessments.

1.1.1.7 Summary of Required Infrastructure Capabilities

To realize this DSAP capability, this effort is developing an underlying software infrastructure that supports C4I planning systems. This enhanced software framework realizes the predictive functionality by supporting simulation capabilities that allow objects to be dynamically created, deleted and reconfigured, while allowing simulations to be calibrated with live data feeds and estimating the state of real-time operations.

The subsequent sections of this Technical Report discuss the implementation of the DSAP software infrastructure using existing simulation and advanced information technology concepts.

- Section 2.0 discusses the Concept of Operations for DSAP and outlines the overarching architecture and design tradeoffs.
- Section 3.0 discusses the design and implementation of a Multiple Replication Framework (MRF) that is at the heart of the DSAP software infrastructure.
- Section 4.0 discusses the effectiveness measurements used to rank plans and alternatives
- Section 5.0 discusses the operation of the MRF.
- Section 6.0 discusses future enhancements to the DSAP infrastructure that will target its deployment on the Global Information Grid (GIG).
- Section 7.0 covers DSAP usage and outlines future work.
- Section 8.0 provides the Bibliography for this Final Report.
- Section 9.0 defines the acronyms used in developing this report.

2.0 DSAP Concept of Operations and Architecture

The Conceptual Operation of the DSAP Infrastructure supports real-time data calibration, real-time state estimation through simulation, and predictive simulation through faster-than-real-time simulation. The concepts of operation for the DSAP Infrastructure are shown in Figure 1 with respect to the Dynamic Situation Assessment capability and Figure 2 with respect to the Prediction capability. In Figure 1, the $x_p(t)$ axis simulates the current plan in real-time. This real-time simulation is used to simulate the results and internal state of the operational picture. As this real-time simulation evolves, it is constantly being updated with Blue and Red Force information from C4I data feeds and databases. These real-time updates, denoted by the $z(t)$ axis, are provided to the real-time simulation of the current plan to calibrate the behavior of the simulated COA. This calibrated real-time simulation is used to estimate the state of the real-time operational picture, $x_e(t)$. This allows us to store the internal state, $x_e(t)$ of the mission in a manner that provides our Dynamic Situation Assessment capability.

Figure 2 illustrates the predictive capability of DSAP. The individual plans, $y(t)$, can be idealized and executed out in time. This basically represents the behavior of the plan when the plan executes as expected. These same plans and their alternatives are then simulated faster-than-real-time, as denoted by the $x(t)$ axis. By executing these plans faster-than-real-time, we provide a predictive look into how a plan and its execution may unfold. Multiple plans and multiple replications of each plan may be executed to provide a statistically significant outlook of a plan's anticipated outcomes based on the current operational information. This provides the Prediction capability.

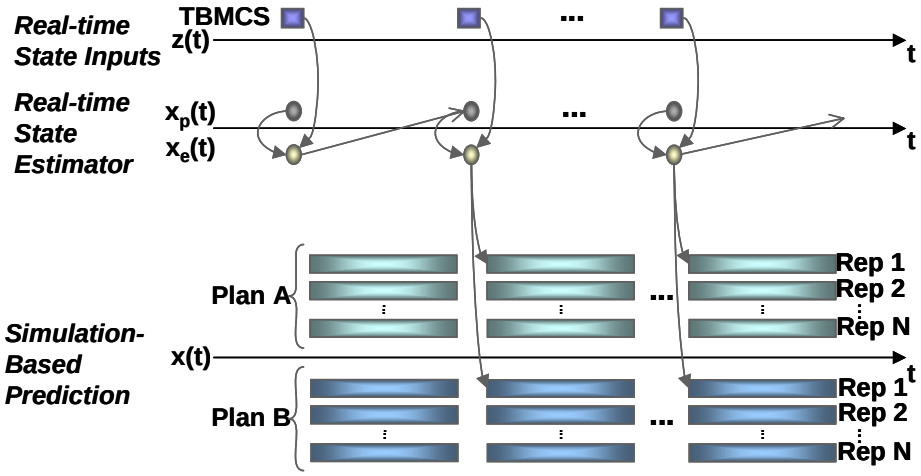


Figure 1: Calibrated Real-time Simulation for Estimation

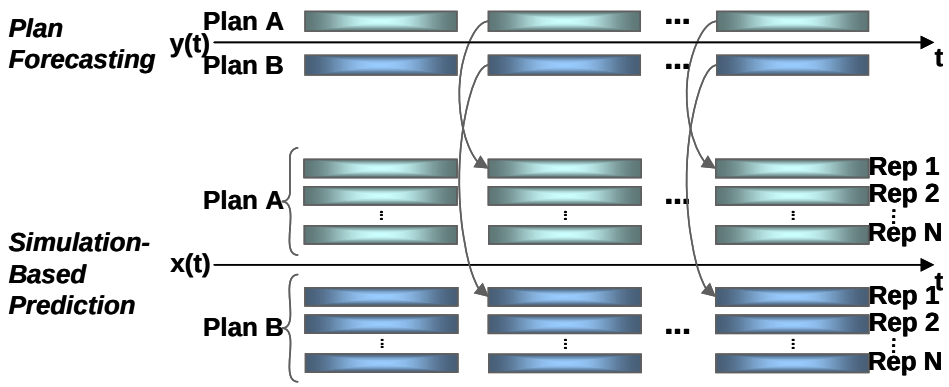


Figure 2: Predictive Simulations Faster-Than-Real-Time

2.1 DSAP Architecture

There are three key components to the DSAP Software Infrastructure: the Multi Replication Framework (MRF), the Optimization Framework, and an Integrated version of Joint SemiAutomated Forces (JSAF) to provide simulation components to the DSAP Infrastructure. The MRF is used to provide a framework for support of multiple replications of any simulation across parallel and distributed platforms. The Optimization Framework is responsible for providing a framework for support of optimization problems across parallel and distributed platforms and the Integrated JSAF allows for performing predictions using the JSAF simulations. The overall architecture is shown in Figure 3. The element in red is the optimization framework, the elements in green are the integrated JSAF, and the elements in black comprise the MRF. The majority of the work performed on this effort covered the MRF and Integrated JSAF.

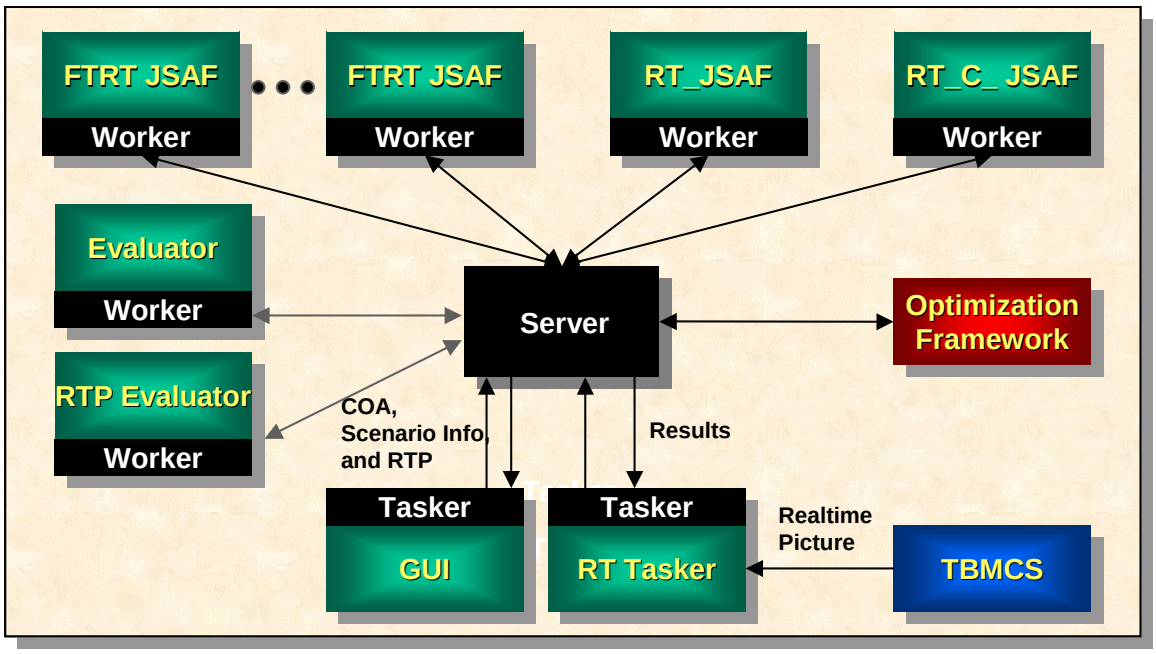


Figure 3: DSAP Software Infrastructure

2.1.1 Multi Replication Framework Operation

The operation of the MRF is detailed in Figure 4. Replications are run on available processing resources across a distributed gird. The results of those replication runs are saved to disk and evaluated. The evaluation is performed to identify the most promising replication modeling an alternate COA. In addition, each replication is evaluated against the real-time picture provided by the Theater Battle Management Core Systems (TBMCS). TBMCS inputs are used to calibrate JSAF running in real-time. This calibration process is used both to calibrate the real-time JSAF simulation to assist in estimating the state of real-world operations, as well as to calibrate predictive simulations and recommend candidates for pruning.

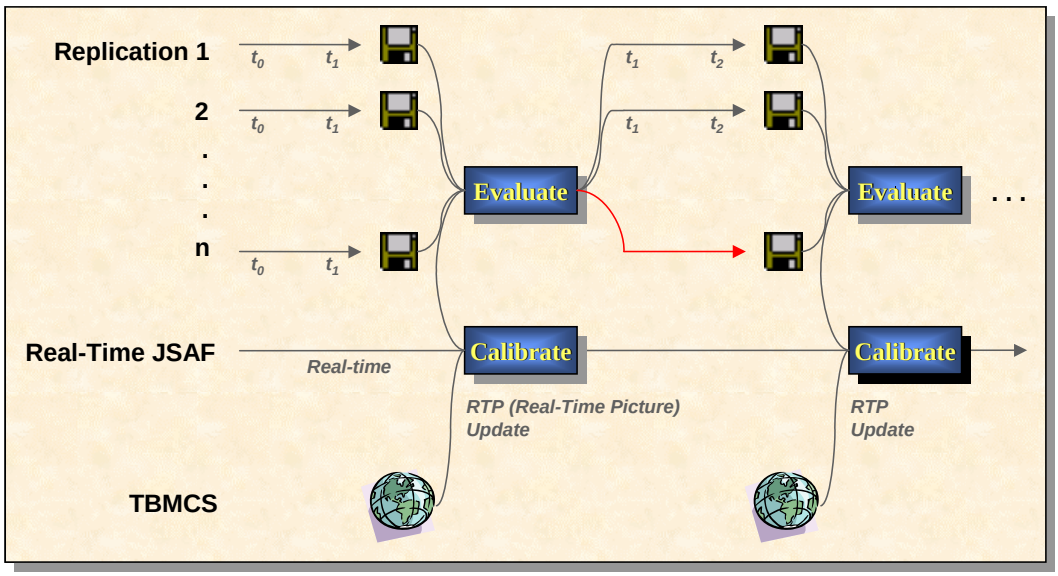


Figure 4: Operation of the MRF

The MRF is built on three key pieces of software infrastructure: Network Communications, a Client Server ORB Infrastructure, and an Extensible Grid Capability. Network Communications supports messaging between platforms supporting the MRF. The Client Server ORB Infrastructure provides the interfaces to simplify the communications between platforms and the Extensible Grid Capability provides the ability to manage the farming of applications in a distributed compute grid. Each of these are described in greater detail in the following subsections.

2.1.1.1 Network Communications

The Network Communications provides highly optimized services to allow single and multiple processor computers to communicate across local and wide area networks. The Network Communications layer minimizes message copying and memory allocation overheads by using free lists to store incoming message parameters of fixed length. The Network Communications also support heterogeneous networks that mix big and little endian data formats. The Network Communications was the most basic building block of advanced information technology used on this effort and served as the basis for both the Client Server ORB and Extensible Grid implementations.

2.1.1.2 Client Server ORB (Object Request Broker) Infrastructure

The Client Server ORB provides simplified interfaces for supporting the communication between platforms. These interfaces have the requirement of providing a distributed object interface to support network programming. These interfaces are language and platform independent. This effort examined three possible solutions for providing this Client Server ORB Infrastructure: ACE + TAO, the Synchronous Parallel Environment for Emulation and Discrete Event Simulation (SPEEDES) / Common Component Simulation Engine (CCSE) ORB Infrastructure, and RAM ORB, an open source ORB infrastructure developed by RAM Laboratories, Inc. in support of another Department of Defense (DoD) effort. A comparison of these solutions is discussed below.

2.1.1.2.1 ACE + TAO

ACE + TAO is middleware that supports portability and software reuse. ACE + TAO is comprised of ACE (Adaptive Communication Environment) and The ACE ORB (TAO). ACE is an open source framework that provides components and patterns for developing high-performance concurrent communication software for distributed real-time and embedded systems. ACE simplifies development of OO (Object-Oriented) network applications and services that utilize interprocess communication, event demultiplexing, explicit dynamic linking, and concurrency. ACE also automates system configuration and reconfiguration and masks operating system differences.

TAO provides an open-source implementation of a Common Object Request Broker Architecture (CORBA) Object Request Broker (ORB). It allows clients to invoke operations on distributed objects without concern for object location, programming language, OS platform, communication protocols and interconnects, and hardware. TAO is built using components and patterns in the ACE framework.

TAO captures key design patterns and optimization principle patterns necessary to develop standards-compliant Quality of Service (QoS)-enabled ORBs. It combines real-time Input/Output (I/O) subsystem architecture and optimization strategies with ORBs to provide vertically integrated ORB end systems supporting end-to-end throughput, latency, jitter, and dependability QoS requirements

2.1.1.2.2 SPEEDES/CCSE Client Server Functionality

SPEEDES/CCSE provides interfaces for a standard communication infrastructure that is used to connect networked simulations together. A general-purpose client/server infrastructure coordinates message passing between machines in a local area network and between multiple local area networks in a wide area network. The client/server infrastructure supports dynamic connectivity to allow new applications to join the system and fault tolerance when applications exit the system. The client/server infrastructure is distributable, redundant, facilitates multiple service types, and coordinates multiple user groups. It also supports heterogeneous networks that mix large and small endian data formats.

A generic client/server model is used to support multiple services types within a server process. Multiple servers may be used to connect local networks to other local networks. In this manner, a spider-web like network of servers tries to minimize message congestion while optimally routing messages. Each server type is represented as a class in the server process. Message headers for

services requested by the client to the server process include information describing the type of service requested, the specific service requested, and the group Id of the requester.

2.1.1.2.3 RAM Object Request Broker

The RAM ORB allows clients to first connect to their server process and then freely send messages to the server. Clients use the RAM ORB layer, then poll to receive incoming messages. Servers differ from clients in that they only process incoming messages. Servers go to sleep when there are no incoming messages to process.

Fault tolerance and dynamic connectivity capabilities allow for new applications to seamlessly join and exit the networked system. The client/server infrastructure supports multiple server types and object Ids.

Multiple servers may be used to connect local networks to other local networks. In this manner, a spider-web like network of servers tries to minimize message congestion while optimally routing messages. Each server type is represented as a class in the server process. Message headers for services requested by the client to the server process include information describing the type of service requested, the specific service requested, and the group Id of the requester.

2.1.1.2.4 Selection of ORB Technology

As a result of our trade study, the RAM ORB was selected for this effort because of its support for two-way interfaces and its support for both networked and shared memory communications, thus allowing for the potential use of multi-processor clients when conducting faster-than-real-time simulation.

2.1.1.3 Extensible Grid Capability

The Extensible Grid works in a server-client implementation consisting of three components that enable maximum performance and highest reliability on a distributed network. The Extensible Grid is built on Object Request Broker (ORB) technology. The components of the Extensible Grid include the Server, Tasker, and Worker, providing the infrastructure necessary to deliver an advanced grid system implementation. Figure 5 shows an Extensible Grid implementation utilizing with the individual components.

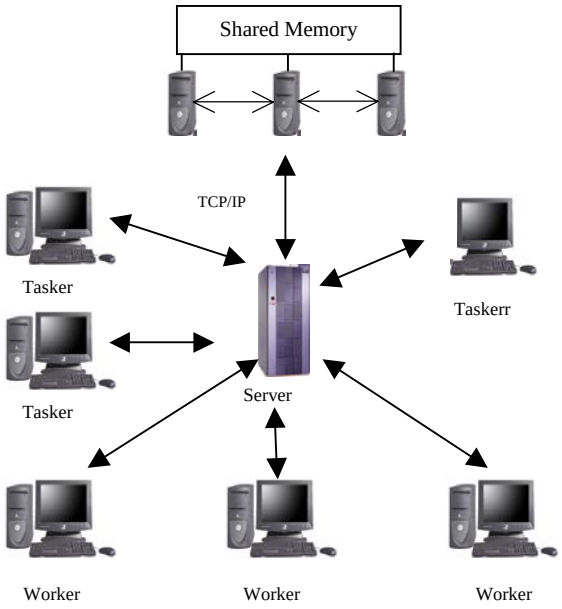


Figure 5: Extensible Grid Implementation

The Server is the backbone of the system and requires no additional implementation on the developer end. The Server is an executable that handles receiving, queuing, and intelligent task distribution. The Server receives individual tasks from the Tasker and distributes those tasks to the receiving Workers.

The Tasker is the main interface for tasking the system. The Tasker's tasks include: 1) instantiating the Tasker and connecting to the Server, 2) sending tasks to the server, 3) waiting for all or some tasks to be completed, and 4) getting the results for each task back from the Server. The Tasker connects to the Server in a ready state, sends tasks and receives results.

As the tasks are being received, the information on a task can be accessed. The task list can be traversed or a specific task can be directly viewed. The Server also informs the Tasker whether a particular task has been addressed by Workers residing on the grid.

The Worker is the component that executes the application on a specific grid node. Just like the Tasker, the Worker connects to the server when it is instantiated. The main focus of the worker is to analyze data received, execute the application, and send back a response.

2.1.2 Optimization Framework

The Optimization Framework feature provides the capability to support optimization of cost functions across parallel and distributed platforms based on the feedback from predictive simulation replications. The Optimization Framework is comprised of a Rollback Framework, a Persistence Framework, Communications, Event Processing, Branch Management Capabilities, a Branch Modeling Framework and a Cost Function Evaluator. Requirements for the Optimization Framework were developed on this effort, however, the design and implementation steps will be addressed further down on the DSAP development roadmap.

2.1.3 Integrating with JSAF

In order to use JSAF as both a predictive tool and as a state estimator, the MRF was integrated with Joint Semi-Automated Forces. A key constraint that was brought about in this area involved

JSAF's use as a predictive tool. To provide a prediction capability, JSAF had to execute faster-than-real-time in order to provide predictive responses to DSAP users operating in real-time. To address this faster-than-real-time constraint, several key functional areas of JSAF must execute in a timely manner. An effort was performed to benchmark the JSAF scenarios to ensure that it could be used as a predictive tool. The benchmarking used an Air Tasking Order from the Korean scenario (korea-ato.xls) that was simulated for over 26 hours of simulation time using JSAF. JSAF was run on a 1.7 GHz Intel i686 processor running RedHat Linux Version 9.0. The platform had a 500 MB hard drive and a 256 KB cache. Several key parameters were benchmarked to ensure that JSAF simulations could be executed within a 15-minute time window. The 15-minute window exists because that is the rate at which TBMCS will be used to calibrate JSAF scenarios running in real-time. The key measurements that were identified through benchmarking included:

- The simulation initialization time needed to be identified.
- The scaling factor for the simulation had to be determined with respect to time.
- The accuracy of the scaled simulation with respect to the real-time scaling factor had to be determined.
- The amount of time required for performing checkpoints and restarts had to be determined.

Benchmarks concerning each of these metrics is discussed in the subsections below.

2.1.3.1 Initialization Time

The initialization time was measured for JSAF simulations for both real-time and scaled real-time simulations. The benchmarking was performed with the GUI turned off. Initialization time was measured with respect to the wall clock. The average initialization time was 2 minutes and 51 seconds for the Korean scenario. Figure 6 depicts the initialization time for the Korean scenario with respect to the real-time scaling factor.

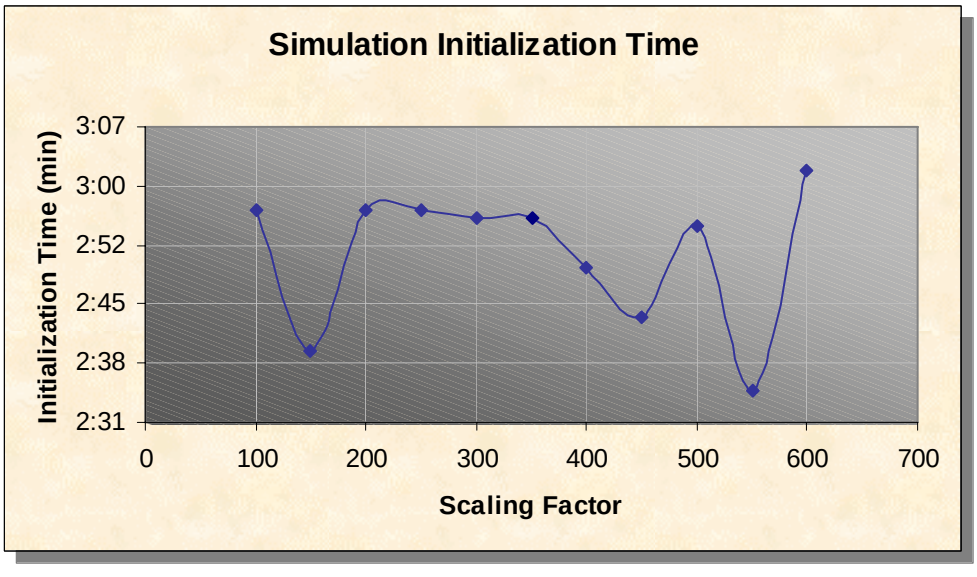


Figure 6: JSAF Initialization Time for Korean Scenario

2.1.3.2 Scaling Factor

The ATO for the Korean scenario was run on JSAF in both real-time, as a scaling factor of real-time, and as fast as possible. Our benchmarks found that the scaling was not linear and that the scenario execution topped out at 350X real-time. The benchmarks of the wall clock simulation time versus the scaling factor are shown in Figure 7. This meant that the fastest simulation execution from start to finish was about 4 and a half minutes.

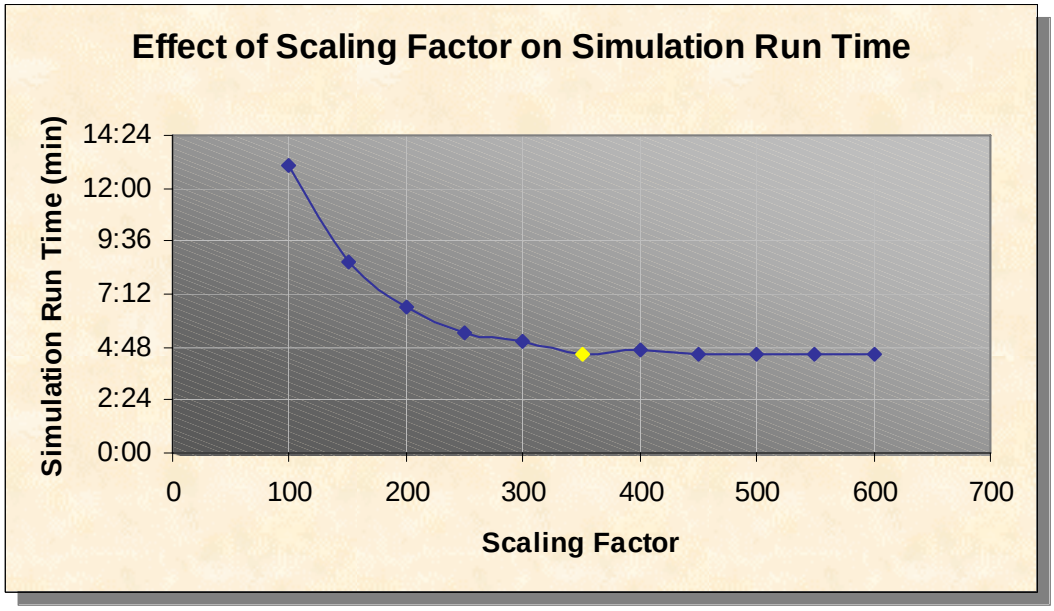


Figure 7: Effect of SimRate on JSAF Execution Time

It should be noted, however, that the simulated JSAF execution at this simulation rate was found to lack accuracy due to events being queued and delayed at the initialization of JSAF executions. This fact rendered excessively large simulation rates useless.

Subsequent simulation runs using DSAP for predictive analysis of JSAF plans have shown that for a modified and substantially smaller version (around 20 entities) of the Korean scenario that RAM Laboratories constructed, a scaling factor of around 14x was the maximum that could be used. The reasons for this are the following: because JSAF is a real-time simulation, the real-time simulation clock immediately starts upon execution or launch of JSAF. When large scaling factors are used, JSAF starts its scaled clock while instantiating the scenario under evaluation. This often pushes events to the right on the timeline while JSAF is constructing entities in the simulation. For large simulations, this scaling factor starts approaching 1x.

2.1.3.3 Checkpoints and Restarts

The Korean scenario was used to benchmark JSAF checkpoints and restarts. No correlation was found between using scaling factors for the simulation rate and the time it took to perform checkpoints or restarts. The average checkpoint time on the platform was 23 seconds. The average restart time was 2 minutes 31 seconds. No restart took longer than 3 minutes 15 seconds to perform.

When DSAP is installed across a distributed grid in a laboratory setting, there were several issues with JSAF surrounding the Checkpoint and Restart process. The major problem found with the

Checkpoint and Restart process is that JSAF often was found to “ghost” checkpointed entities across a restarted simulation. This meant that JSAF often detected other running JSAF executions on the network and instantiated those entities as well as its own. In addition, there were issues with constructing Red Force target data when running more than one JSAF execution.

2.1.3.4 JSAF Integration

This subsection covers the effort required for integrating JSAF with the MRF for use as both a predictive tool and a state estimator. Specifically, the MRF needs to load scenarios, save scenarios, execute scenarios and adjust the simulation rate. All of these must be executed from a command line to trigger JSAF execution.

2.1.3.4.1 Loading and Saving JSAF Scenarios

The following code snippets detail the process for loading and saving JSAF scenarios. The loading and saving of JSAF scenarios allows for checkpointing and restarting JSAF simulations. The restart process will serve as the starting point for branching replications.

Code Segment 1: Loading and Saving JSAF Scenarios: Loading and Saving JSAF Scenarios

libpo.h

```
// Save all objects into a file
extern int32 po_begin_save(PO_DATABASE *db,
                           char        *fname);
// Load a file containing persistent objects. 'prevent_foreign_origin' is a
boolean flag
// that indicates wether or not libpo should fail if the saved scenario's
creator (ie.
// hostname of the GUI that saved the scenario) doesn't match the hostname of
the GUI that
// is trying to load it.

extern int32 po_begin_load(PO_DATABASE *db,
                           char        *fname,
                           int32        prevent_foreign_origin = 0);
// Completly clear out a database

extern void po_new_scenario(PO_DATABASE *db);
```

libpo_local.h

```
// po_write.c:

extern int32 po_save_to_file(PO_DATABASE *db,
                             char        *fname);
```

2.1.3.4.2 Setting SimRate and Processing Up to a Given Time

The following code snippets detail the process for setting the SimRate to run as fast as possible and to process up to a specified time. These capabilities can be used to run JSAF faster than real time and will allow JSAF to be used in a predictive capacity.

Code Segment 2: Setting the SimRate and Processing Up To A Given Time

libsched.h

```
// Sets whether to keep time_realtime_clock slaved to real time, or whether
to skip ahead to
// the next time there's something to do.

extern void sched_set_fast_as_possible(int32 run_fast_as_possible);


// Runs the scheduler until the specified deadline.  The time used is that
from
// time_realtime_clock.  A deadline of 0xFFFFFFFF will run indefinitely.  To
just run
// pending functions, pass time_last_realtime_clock + 1.

extern void sched_invoke_functions_until(uint32 deadline);
```

2.1.3.4.3 Adjusting SimRate, Pausing Simulation, and Extracting Simulation Time

The following code snippet details the process for adjusting the SimRate, pausing the simulation and obtaining the simulation time. These features are necessary to calibrate the running JSAF simulation and to extract run-time diagnostics.

Code Segment 3: Adjusting the SimRate, Pausing the Simulation, Extracting the Simulation Time

libtime.h

```
// Sets the relationship between the simulation clock and real time.

extern void time_set_simulation_rate(float64  rate);


// Gets the simulation rate.

extern float64 time_get_simulation_rate(void);


// Stops advance of simulation time.  If multiple pause handles are
outstanding, the
// simulation will be paused if ANY of them request a pause.

extern void time_pause(TIME_PAUSE_HANDLE handle);
```

```
// Starts advance of simulation time.  If multiple pause handles are
outstanding, the
// simulation will not resume until ALL of them are unpaused.

extern void time_unpause(TIME_PAUSE_HANDLE handle);


// Returns the current simulation time.
```

```
extern uint32 time_simulation_time(void);


// Returns the realtime at the last simulation frame

extern uint32 time_realtime_last_sim_frame(void);


// Returns the time of the real time clock in milliseconds.

extern uint32 time_realtime_clock(void);


// Returns the time at the last check of the realtime clock.
extern uint32 time_get_last_realtime(void);
```

2.1.3.4.4 Loading Spreadsheets and Saving Parameters

The following code snippet demonstrates the process for loading spreadsheets into JSAF and saving parameters. This process is used to save a checkpointed simulation for use in determining the cost or goodness of that simulation.

Code Segment 4: Loading and Saving Spreadsheets

libspreadsheet.h

```
extern void sprdsht_load_spreadsheet(char        *filename,
                                     PO_DATABASE *po_db);


extern void sprdsht_read_sprdsht(FILE        *readfile,
                                 char        *fname,
                                 PO_DATABASE *db);


extern void sprdsht_write_pts_to_file(FILE        *write_file,
                                      char        *fname,
                                      PO_DATABASE *db,
```

```
                                    PO_DB_ENTRY *ovl_entry,
                                    uint8        save_type);

extern void sprdsht_write_units_to_file(FILE        *write_file,
                                        char        *fname,
                                        PO_DATABASE *db,
                                        uint8        save_type);

extern void sprdsht_write_lat_long(FILE    *file_ptr,
                                   float64 gcsloc[XYZC]);

// Converts and calls the above

extern void sprdsht_write_lat_long(FILE           *file_ptr,
                                   PointLocation3D *pt);

// Converts and calls the above

extern void sprdsht_write_lat_long_3e(FILE    *file_ptr,
                                      float64  xval,
                                      float64  yval,
                                      float64  zval,
                                      int32    cell);

// Write points in MGRS coordinates

extern void sprdsht_write_mgrs_coords(FILE    *file_ptr,
                                      float64  gcsloc[XYZC]);

// Converts and calls the above

extern void sprdsht_write_mgrs_coords(FILE           *file_ptr,
                                      PointLocation3D *pt);

extern void sprdsht_write_mgrs_coords_3e(FILE    *file_ptr,
                                         float64  xval,
                                         float64  yval,
                                         float64  zval,
                                         int32    cell);
```

```
// Write points in GCC coordinates

extern void sprdsht_write_gcc_coords(FILE    *file_ptr,
                                     float64  gcsloc[XYZC]);

// Converts and calls the above

extern void sprdsht_write_gcc_coords(FILE            *file_ptr,
                                     PointLocation3D *pt);
```

2.1.3.4.5 Example

The follow code snippets demonstrate how the original JSAF code (Code Segment 5) can be modified to support the loading and executing of multiple replications.

Code Segment 5: Original JSAF Code

```
main.c // JSAF original example

int main(int argc, argv_t argv) {

  int status = main_init(argc, argv);

  if (status)
    return status;

  // Fire up the scheduler

  sched_invoke_functions_until(0xFFFFFFFF);

  // Not reached

  return(0);

}
```

Code Segment 6: Modifications For Integration with MRF

```
main.c // RamLabs mod

int main(int argc, argv_t argv) {

  int status = main_init(argc, argv);

  if (status)
    return status;

  // run 10 batches

  int counter;
  for (counter=0; counter<10; counter++) {
    cout << "Counter = " << counter << endl;

    // have correct scenario path

    char *name = "/users/jsaf/JSAF5/scenarios/scenarioName.1/scenarioName";
    cout << "Loading scenario: " << name << endl;

    // load scenario

    po_begin_load(static_po_db, name);
    cout << "Done loading scenario: " << name << endl;

    int time;
    for (time=0; time<10000; time+=1000) {
      cout << "Process up to " << time << endl;

      // Fire up the scheduler

      sched_invoke_functions_until(time);
    }

    po_new_scenario(static_po_db);
  }

  cout << "Done - exiting..." << endl;
```

```
  // Not reached
  return(0);

}
```

3.0 Implementing the DSAP MRF

An overview of the current capabilities of the DSAP prototype is shown via the MRF in Figure 8. The MRF serves to farm-out and run multiple replications of plans and alternative COAs via faster-than-real-time simulation. When real-time updates are available, state information from the real-time state estimation simulation are calibrated with real-time C4I inputs, saved and compared with the state information from the predictive plans. Plan replications that diverge from the real-time picture beyond a prediefined threshold are automatically pruned and replaced. Our MRF prototype manages this entire process by utilizing TBMCS for our real-time C4I inputs and JSAF as both the real-time and faster-than-real-time simulation components. JSAF was selected as our simulation component because of its ability to simulate a Joint Urban Operations (JUO) environment (as well as theater operations) as well as its enhanced support for intelligent ground clutter models. It should also be noted that other simulations can be used as the simulation component depending on the desired application.

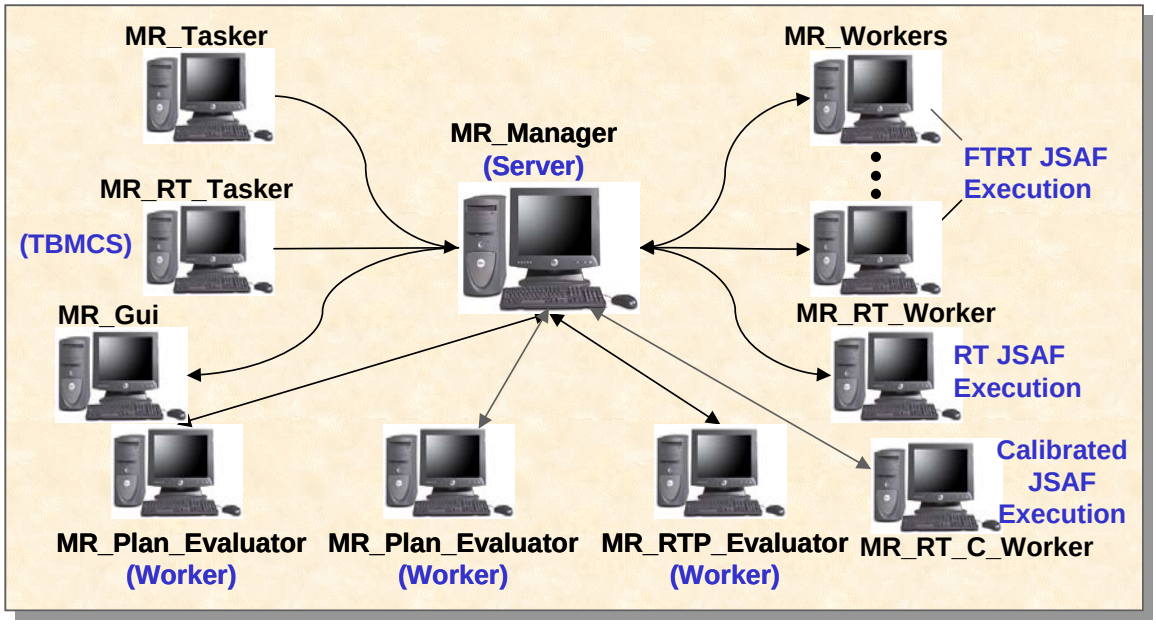


Figure 8: The MRF

Currently, the MRF has the capability to simultaneously simulate multiple plans and replications of a plan faster-than-real-time across a network of computers. The MRF has the ability to compare the simulation state of a faster-than-real-time replication with the plan objectives and real-time picture. Replications that diverge from the real-time picture beyond a predefined threshold are automatically pruned and replaced; Commanders have the capability to manually prune ineffective plans and task alternative plans. The MRF runs a real-time state estimation simulation that can pull data from live sources; this data is used to continuously update the real-time simulation. The components required to provide this functionality are described in the following subsections.

3.1 MRF Components

The MRF contains three basic types of components: taskers, workers, and a manager. Taskers function to task the manager with applications for workers to run, and specify how these

applications should be initialized. Workers execute the tasks assigned by the manager, including executing the simulation replications, saving the results of the replications, and evaluating and comparing the results of the replications to the plan objectives and real-time picture. The manager divides the replications into smaller time segments and assigns these tasks to the workers, handles the bookkeeping, and tackles flow control issues. Figure 8 illustrates each of these components and their connectivity with the manager. The role of each of the specific MRF components is discussed in the following subsections.

3.1.1 Tasker

The Tasker component interfaces with Command and Control to send a predictive simulation task to the server. The Tasker provides connectivity to the server and allows the user to specify initialization parameters such as the simulation execution name, initial scenario file, start and end time, simulation scaling rate, and replication number.

3.1.2 Real-Time Tasker

The Real-Time Tasker component issues a task to the server to initiate the real-time worker. The Real-Time Tasker provides connectivity to the server and allows the user to specify the simulation execution name, initial scenario file, name of the plan the task corresponds to, and the time interval between saving the state of the simulation.

The Real-Time Tasker component, MR_TBMCS_Tasker, provides the capability to allow the user to retrieve the Real-Time Picture (RTP) from TBMCS or another C4I data source via command line or GUI. The sequence diagram defining the operation of the MR_TBMCS_Tasker is shown in Figure 9.

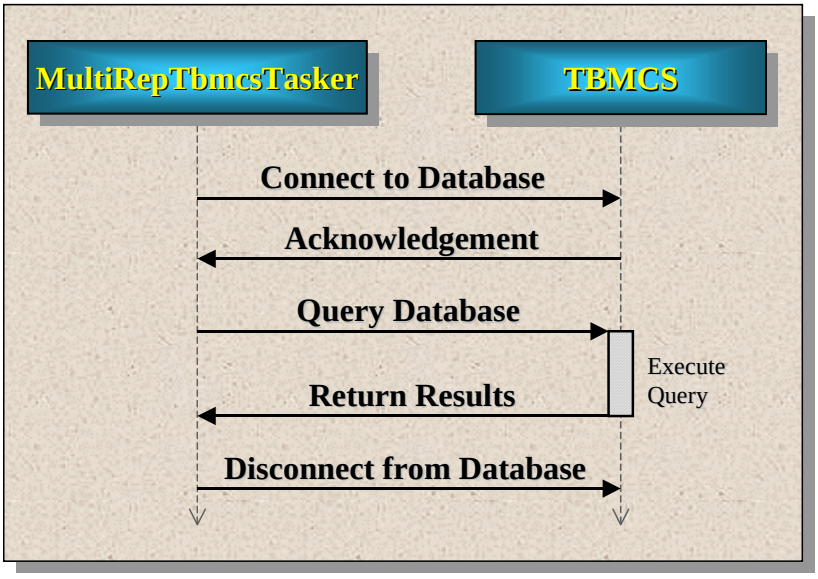


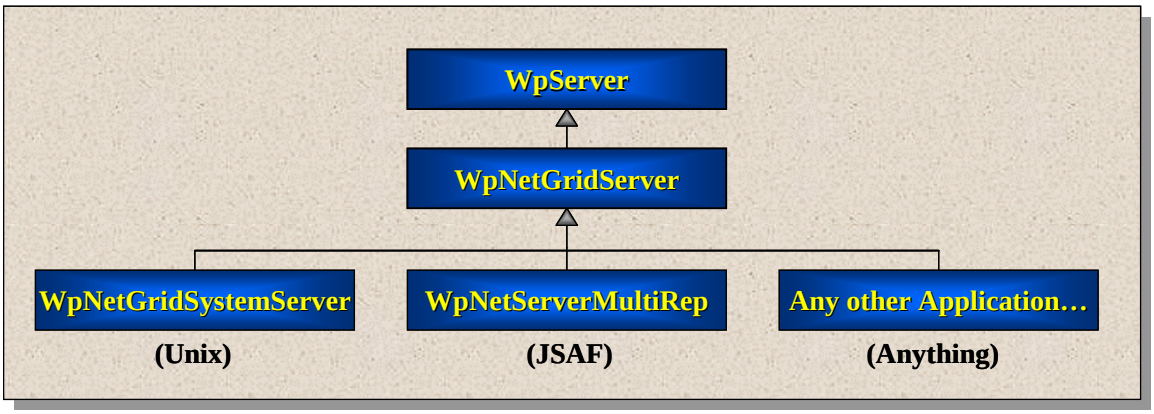
Figure 9: Sequence Diagram for MR_TBMCS_Tasker and TBMCS:

The MR_TBMCS_Tasker is a Tasker component that automates the process of retrieving real-time C4I information from TBMCS. For TBMCS connectivity, both ODBC and JDBC were tried. Some problems existed with ODBC and have not been resolved. The Tasker/JDBC approach to TBMCS connectivity has been implemented and tested.

3.1.3 Manager

The server, or manager, component is the core of the MRF. The server is responsible for 1) managing the execution of long replications by splicing them in time, 2) constructing the necessary parameters needed for a worker to launch and save a JSAF execution, 3) constructing the necessary parameters for launching an evaluation on an evaluator component, 4) displaying diagnostics related to the execution of multiple replications, 5) identifying when replications are completed, 6) pruning and re-tasking replications that are off course from the real-time picture, and 7) restarting unfinished replications in the event of a worker crash or disconnect.

The Server design builds off of the Server used to implement the Extensible Grid. The Server capability for this effort inherits from the WpNetGridServer, which is the Server for the Extensible Grid, which in turn inherits from WpServer, which is the basic server capability in the



WarpIV Framework. The UML Class Diagram for the Server design is shown in Figure 10.

Figure 10: UML Diagram for Server Component

The Server is responsible for implementing the following functionality:

- Managing executions of long replications by splicing them in time.
- Constructing the necessary parameters needed for launching an execution on a Worker.
- Displaying diagnostics related to the execution of multiple replications.
- Supporting multiple groups for the same exercise on the same server.
- Providing a measure of fault tolerance for running multiple replications.
- Restarting unfinished replications if a Worker crashes or is unable to complete its task.

The Server component, MR_Manager, is responsible for managing the execution and evaluation of the replications. The Activity Diagram for the MR_Manager with respect to the *ProcessRtp()* function is shown in Figure 11. This Activity Diagram defines the process for managing the replications through their RTP evaluation.

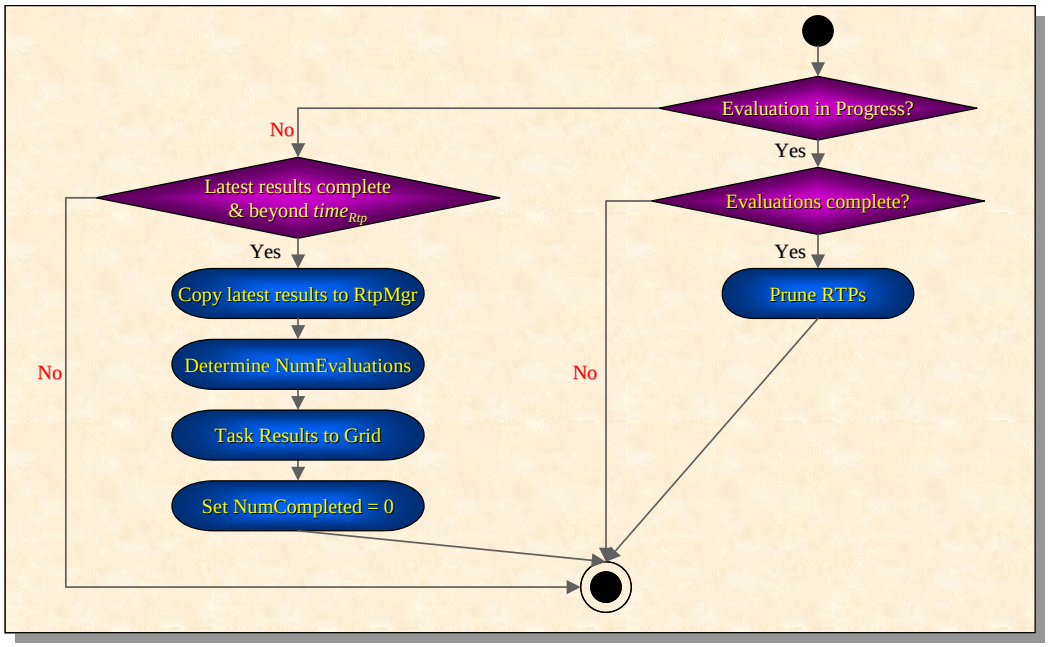


Figure 11: Activity Diagram for MR_Manager's ProcessRtp()

3.1.4 MR_Console and Graphical User Interface (GUI)

The MR_Console component provides the user with diagnostics with respect to operation of the MRF. The MR_Console also allows the user to monitor the status of the MRF. The Sequence Diagram for the MR_Console is shown in Figure 12. In addition to simply monitoring status and setting the time interval for faster-than-real-time simulations, the MR_Console has been modified to host our Graphical User Interface. The MR_Console now queries the server to return the status of replications, provides functionality to modify time intervals and end times, provides functionality to modify pruning thresholds, and provides the capability to allow the user to prune replications or plans using the GUI.

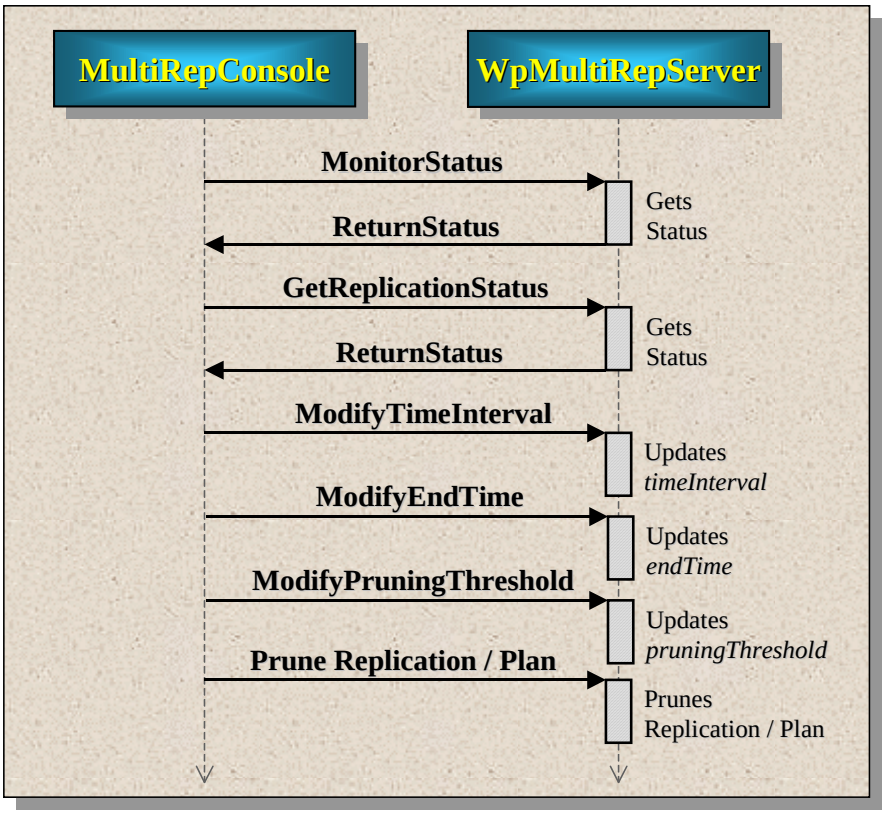


Figure 12: MR_Console and GUI Control Flow

The GUI provides a graphical user interface component for MRF users. The GUI provides a wrapper to interface with the tasker and server components that enable the user to kick off tasks, modify the simulation end time, modify the pruning threshold, prune replications, prune plans, and visualize the relative effectiveness of each plan.

3.1.5 Worker

The Worker component, MR_Worker, receives simulation tasking from the server and launches predictive JSAF executions that run faster-than-real-time. The Worker is responsible for launching JSAF replications faster-than-real-time for a predetermined length of time. The Worker receives a command from the MR_Manager to launch a JSAF execution. This command is accompanied by parameter sets (specifying the SimRate, start time, end time and other variables), environment variables and scenario spreadsheets. The Worker also gathers the results from the replication execution in spreadsheet format and sends the information back to the MR_Manager.

Upon completion of the replication, the Worker saves the state of the simulation to disk and sends it to the server for later evaluation and comparison with real-time data and the plan objectives. Replications that stray from the real-time picture beyond a predefined threshold are automatically pruned, re-tasked by the server, and initialized to match the current state. Replications that fail to meet the plan objectives can be manually pruned by Command Staff, and if pruned, they are automatically re-tasked by the server and initialized to match the current state.

The Sequence Diagram specifying the operation of the Worker executing FTRT JSAF scenarios is shown in Figure 13 with respect to the rest of the MRF.

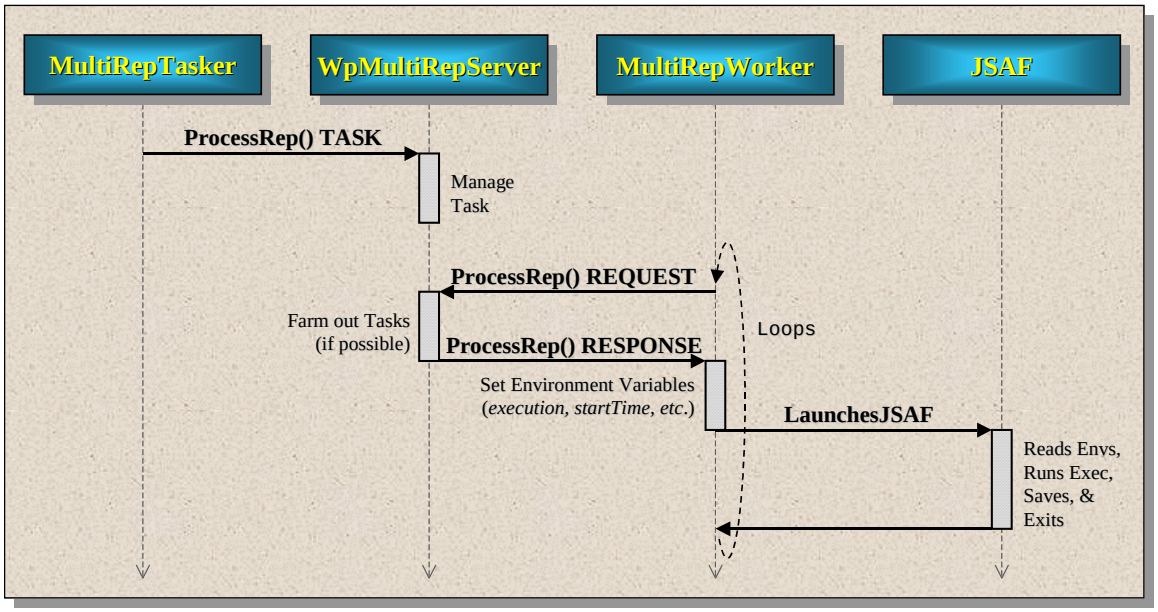


Figure 13: MR_Worker Control Flow for Executing Faster-Than-Real-Time Simulations

3.1.6 Real-Time Worker

The Real-Time Worker component, MR_RT_Worker, receives a simulation task from the server to launch a JSAF execution in real-time. The MR_RT_Worker is used to run real-time JSAF executions (or executions of other simulations) in order to support the state estimation capability in the MRF. The MR_RT_Worker is responsible for running the real-time simulation, updating the simulation with TBMCS information (in the case of the calibrated real-time simulation), and saving checkpoints of the simulation to intermediate results files for effectiveness evaluations. The sequence diagram of the MR_RT_Worker is shown in Figure 14. For the MR_RT_Worker, the worker connects to the server and requests tasking. The Tasking, when provided to the server from the MR_RT_Tasker (the old MR_TBMCS_Tasker) then assigns tasks to the Server, which are passed to the MR_RT_Worker. The MR_RT_Worker executes JSAF tasks and sends results back to the Server every 15 minutes. In addition, the MR_RT_Tasker continually updates the real-time simulation every 15 minutes.

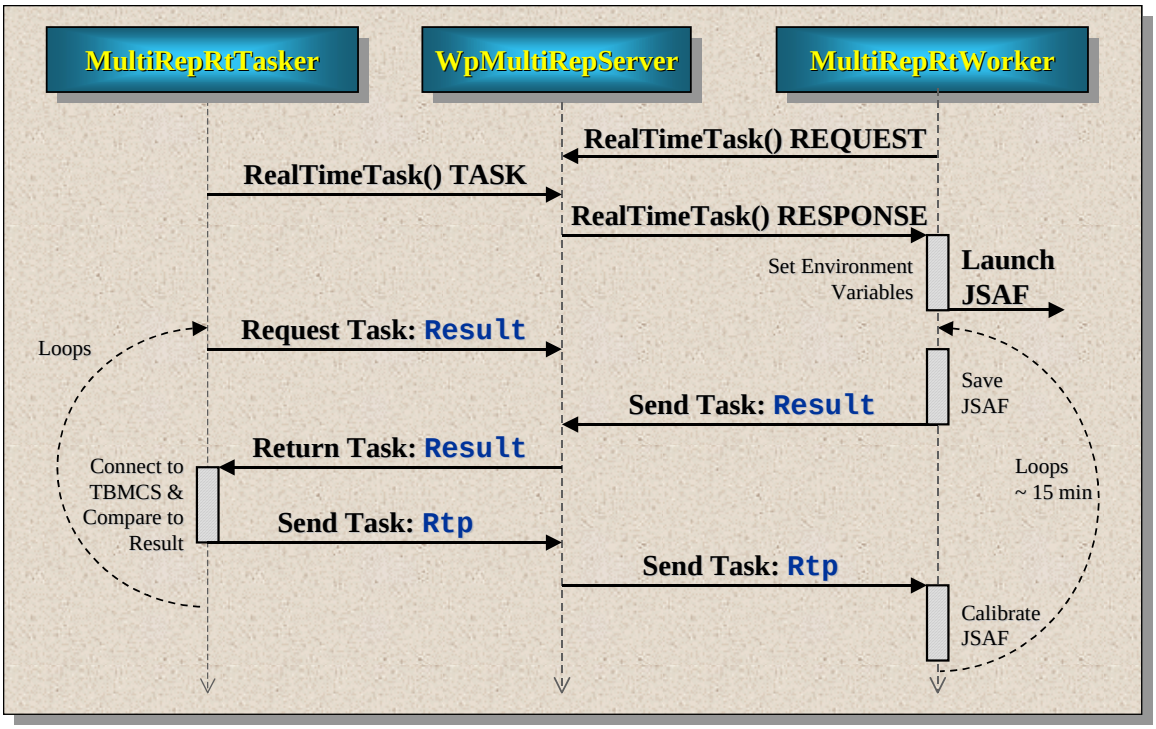


Figure 14: Control Flow for MR_RT_Worker

3.1.7 Plan Evaluator

The Plan Evaluator component, MR_PlanEvaluator, is responsible for comparing the state of the saved faster-than-real-time replications with the plan objectives. The MR_PlanEvaluator is a Worker that evaluates the results of the JSAF replication executions against other results. The MR_PlanEvaluator takes each of the result spreadsheets and evaluates them to determine the "best" plan. The evaluation is performed by executing the function *PlanEvaulator()*. The control flow for the MR_PlanEvaluator is shown in Figure 15, when considering the sequence of operations between the MR_PlanEvaluator, MR_Server, and MR_Worker. The MR_PlanEvaluator requests tasks from the server. When results spreadsheets are available at the MR_Server, those results are tasked to the MR_PlanEvaluator, which evaluates the effectiveness of each plan. The effectiveness results are then sent back to the Server, and the MR_Evaluator is also tasked to begin evaluatoing those results against the current real-time picture.

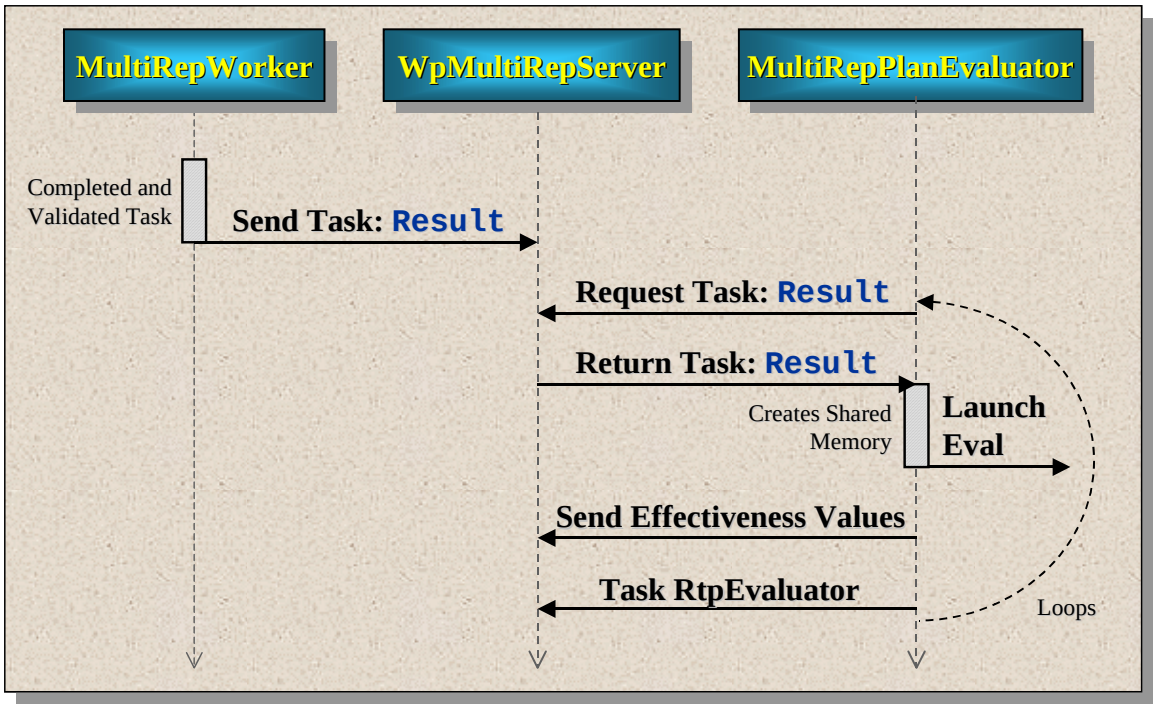


Figure 15: Control Flow for MR_PlanEvaluator

3.1.8 Real-Time Picture Evaluator

The Real-Time Picture (RTP) Evaluator component, MR_RTP_Evaluator, is responsible for comparing the state of the saved faster-than-real-time and real-time simulations with the real-time picture. It is important that replications projecting into the future match reality at the time of the real-time update. Replications that noticeably diverge from the real-time picture are automatically pruned, re-tasked by the server, and initialized to match the current state. The RTP evaluator launches an evaluation program that performs the actual analysis by applying a weighting function that ultimately generates a relative effectiveness value. The server compares this value with a predefined threshold to determine whether the simulation is pruned.
The MR_RTP_Evaluator is a Worker that evaluates the results of the JSAF replication executions against the updated real-time picture. The MR_RTP_Evaluator takes each of the result spreadsheets and evaluates them against the new real-time information. If the evaluation process deems that certain replications are no longer valid when compared to the RTP, then those replications are pruned and new replications will be initiated in their place.

A sequence diagram for the operation of the MR_RTP_Evaluator with respect to other elements of the MRF is shown in Figure 16 and its corresponding state diagram is shown in Figure 17. In these diagrams, the MR_RTP_Evaluator receives both the RTP and Results from the MR_Server. In addition, the control flow provides functionality for looping through RTP evaluation results, and pruning replications that exceed some (user-specified) predefined threshold. This control flow also assumes that the MR_RTP_Evaluator has connected to the server.

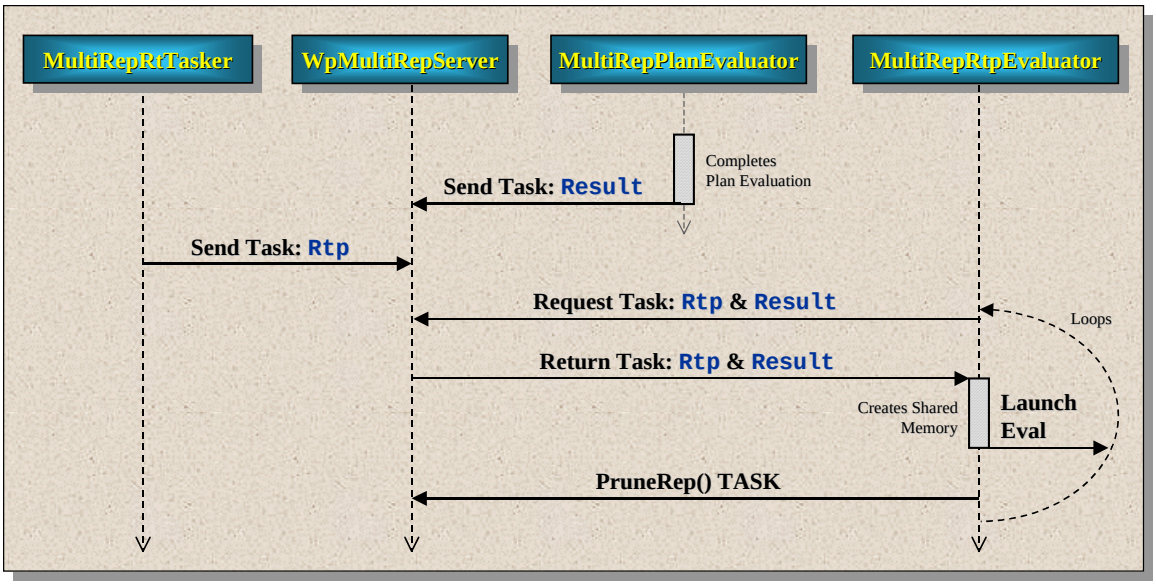


Figure 16: Control Flow for RTP Evaluator

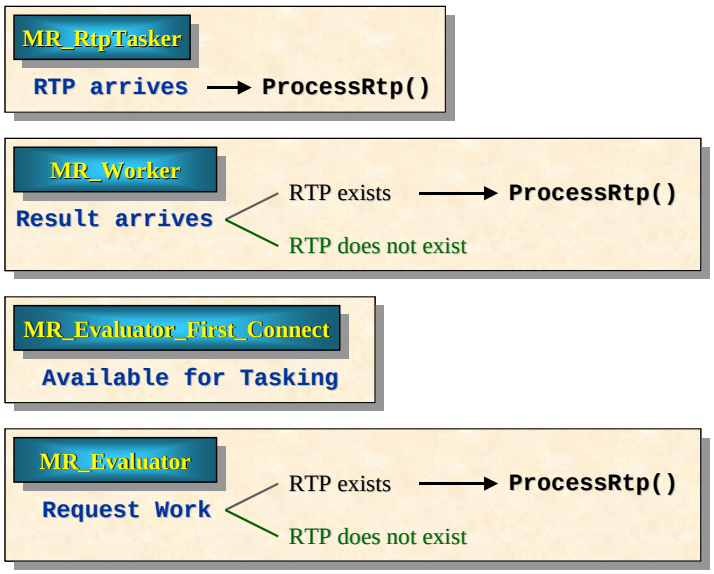


Figure 17: State Diagram for the RTP Evaluator

3.2 Overall Control Flow for Predictive Operations

Each of the preceding subsections has detailed the operation of the MRF for running predictive analysis of plans while calibrating and evaluating the results with real-time-picture information extracted from TBMCS. Sequence diagrams in these subsections have presented a localized view of the flow of control between different components of the MRF system. A key element in ranking plans and alternatives is the calculation of effectiveness metrics. The theory behind these calculations is presented in Section 4.0. A comprehensive view of this control flow for predictive operations is shown in Figure 18.

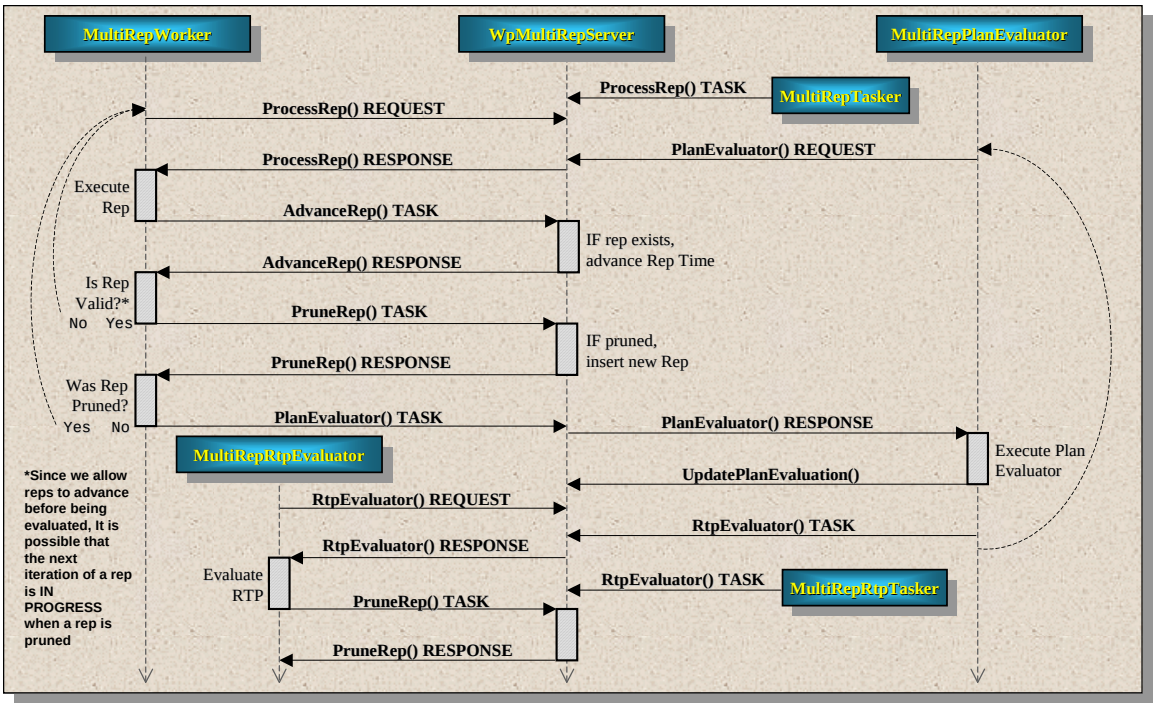


Figure 18: Control Flow for Predictive Operations

4.0 Effectiveness Calculations

This section describes the two unique measures of effectiveness, Raw Effectiveness and Relative Effectiveness, used to rank plans and identify candidates for pruning.

4.1 Raw Effectiveness Defined

Raw effectiveness is a value between zero and one indicating the overall effectiveness of the plan. A value of one indicates complete success (i.e., all Red entities/assets destroyed and no Blue entities/assets destroyed). A value of zero indicates complete failure (i.e., all Blue entities/assets destroyed and no Red entities/assets destroyed). The raw effectiveness is not an indication of how accurately the plan has been followed; rather it is a measure of the overall outcome.

4.1.1 Calculating Raw Effectiveness

The intrinsic value for each entity (or asset) in the battle can be defined at a given point in time. Normally, the intrinsic value does not change for each entity. However, it is possible for new entities to enter the battle, or for new information to be provided indicating changes to the intrinsic value of an entity.

$I_i^B(t)$ = Intrinsic value for blue *entity*$_i$ at time *t*

$I_i^R(t)$ = Intrinsic value for red *entity*$_i$ at time *t*

The actual value for each entity (or asset) in the battle can be defined at a given point in time. The actual value ranges from zero (meaning that the entity/asset has been destroyed) to the intrinsic value (meaning that the entity/asset has perfect health, has not diminished its capacity to engage in battle, and has a full fuel supply).

$A_i^B(t)$ = Actual value for blue *entity*$_i$ at time *t*

$A_i^R(t)$ = Actual value for red *entity*$_i$ at time *t*

The total intrinsic values for Blue and Red entities/assets in the battle can be specified at a given point in time:

$$I^B(t) = \sum_i^{N_B} I_i^B(t)$$

$$I^R(t) = \sum_i^{N_R} I_i^R(t)$$

The total actual values for Blue and Red entities/assets in the battle can be specified at a given point in time:

$$A^B(t) = \sum_i^{N_B} A_i^B(t)$$

$$A^R(t) = \sum_i^{N_R} A_i^R(t)$$

Utilizing these equations, the Raw Effectiveness can be calculated by the following equation:

$$RawEffectiveness(t) = \frac{A^B(t) + \left[I^R(t) - A^R(t)\right]}{I^B(t) + I^R(t)}$$

A successful plan normally results in the raw effectiveness value increasing over time. However, it is possible for a plan to accomplish its mission, even though the raw effectiveness decreases. This would happen if the cost of completing a mission turns out to be higher than the overall gain.

Because of the uncertain nature of predicting the outcome of plans, it is important to execute multiple replications and statistically analyze the results. The mean and standard deviation are provided for each time step in the simulation. These mean values and their standard deviations can be fitted using χ^2 analysis to obtain time-based curves that can provide trend analysis.

Thus, the raw effectiveness can be thought of as the overall measure of effectiveness of the plan. This is different from the relative effectiveness, which is a measure of the plan performance (i.e., how accurately the plan was followed).

4.2 Relative Effectiveness Defined

The relative effectiveness is a measure of the plan performance, indicating how accurately the plan was followed. Two types of relative effectiveness are computed: (1) simulated projections in time vs. plan expectations from the ATO, and (2) simulated projections at the current time with respect to the real-time picture.

In the first case, the relative effectiveness provides a prediction of the uncertainty of the plan performance over time. It predicts when the plan might fall apart, and when new planning may be required. It provides insight on the chaos that may ensue during the fog of war. Multiple replications per plan are required to determine the anticipated outcome. Some replications may deviate from the plan due to statistical variances or uncertainties in the planned scenario, while other replications produce the anticipated outcome. A statistical analysis of the relative effectiveness is used to help characterize the plan dispersion.

In the second case, the relative effectiveness allows simulation replications that began their execution in the past to verify that they match the current real-time picture. The relative effectiveness for this second case is used to prune those replications that do not match the real world picture. New simulation replications are restarted to replace those replications that were pruned using the real-time picture entity states to initialize the simulation.

4.2.1 Calculating Relative Effectiveness

The state of each entity/asset in the plan at any point in time can be defined as an abstract vector of values. These state values can be projected forward in time either from simulation or from the actual planned expectations directly from the Air Tasking Order (ATO). State values can also be provided through the real-time picture with live data feeds that are provided into the system. These three kinds of state vectors are defined below.

The simulation projects the state vectors for each entity/asset in the scenario.

$\vec{X}_i^B(t)$ = Simulated state vector of blue *entity*$_i$ at time *t*

$\vec{X}_i^R(t)$ = Simulated state vector of red *entity*$_i$ at time *t*

The expected state vectors for each entity/asset in the plan is projected from the ATO to predict the outcome of the plan.

$\vec{Y}_i^B(t)$ = Projected state vector of blue *entity*$_i$ at time *t*

$\vec{Y}_i^R(t)$ = Projected state vector of red *entity*$_i$ at time *t*

The real-time picture state vector is defined as follows.

$\vec{Z}_i^B(t)$ = Observed state vector of blue *entity*$_i$ at current time *t*

$\vec{Z}_i^R(t)$ = Observed state vector of red *entity*$_i$ at current time *t*

These vectors can be used to calculate the two cases for calculating the Relative Effectiveness for simulated projections and through the use of the real-time picture. The calculation of Relative Effectiveness for these two cases is shown below:

Case 1 – Simulated projections: $RelativeEffectiveness(t) = 1 - \left(\frac{1}{I^B + I^R}\right)\sum_i^N \left|\vec{X}_i(t) - \vec{Y}_i(t)\right|$

Case 2 – Real-time picture: $RelativeEffectiveness(t) = 1 - \left(\frac{1}{I^B + I^R}\right)\sum_i^N \left|\vec{X}_i(t) - \vec{Z}_i(t)\right|$

Note that for each of these cases, the magnitude of the vector difference is weighted for each vector value. The overall magnitude for each term in the sum is normalized and lies between zero and $I^i(t)$.

A score of one would indicate that the plan is being executed as expected. A score of zero would indicate complete chaos, meaning that the plan has fallen apart and is no longer valid. For Case 2, this would indicate that the plan does not agree with reality and should be pruned/restarted.

5.0 Operating the MRF

The MRF GUI is shown in Figure 19. The GUI provides a graphical interactive interface to the MRF that allows Commanders to task the execution of simulation plans and replications, view the progress and performance of the plans, and prune ineffective plans. The graph at the bottom of the GUI plots the raw and relative effectiveness of each plan over time. These metrics are used to gauge the effectiveness and performance of the Commander's plan.

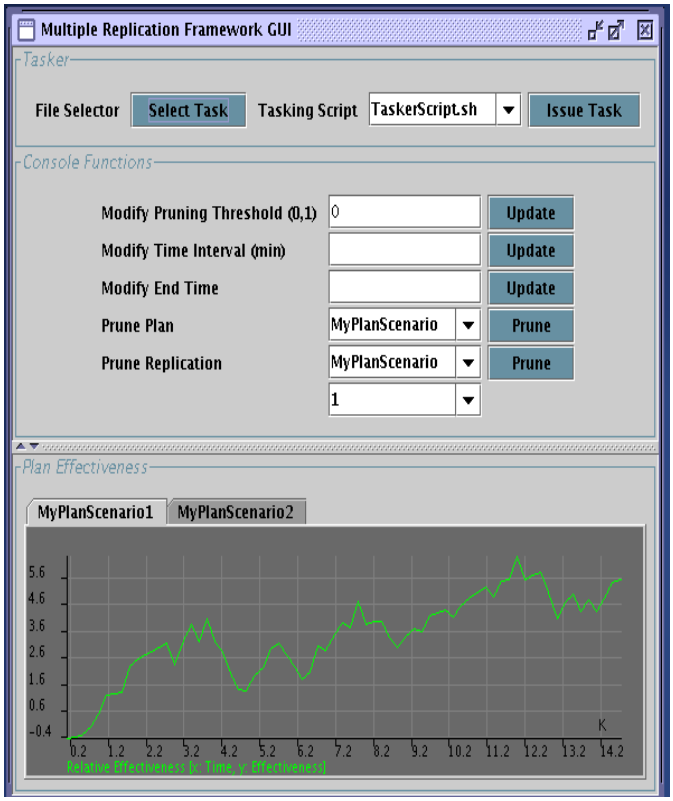


Figure 19: The MRF GUI

Plans and replications are initiated in the MRF by issuing a tasking script from the GUI. As shown in Figure 20, a file selector tool allows the Commander to select a tasking script to kick off the process. The GUI will be expanded to allow the Commander to start plans and replications via menus instead of scripts.

After the script has been selected, the MRF takes control by sending the task to available workers, as shown in Figure 21.

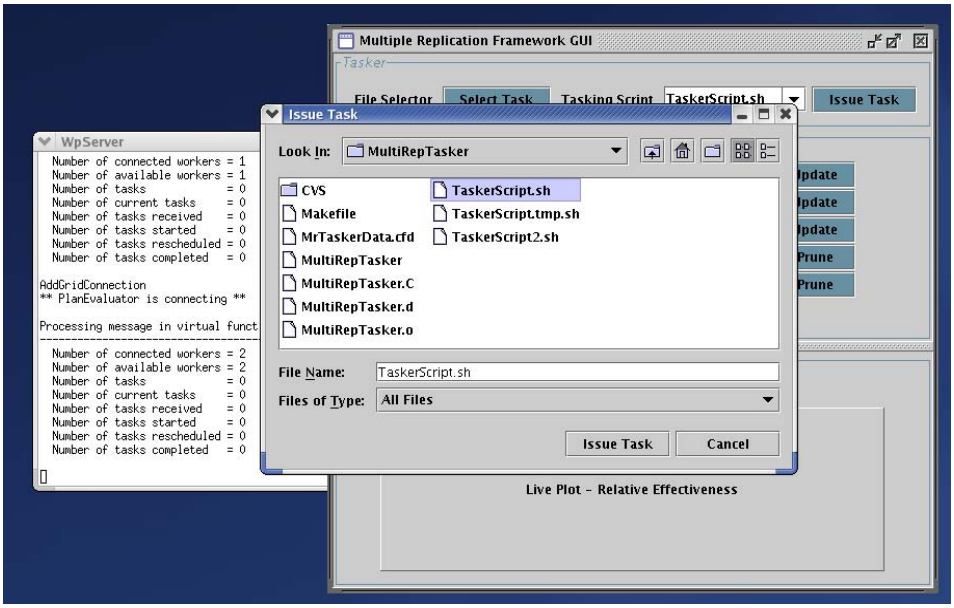


Figure 20: Selecting and Issuing a Tasking Script

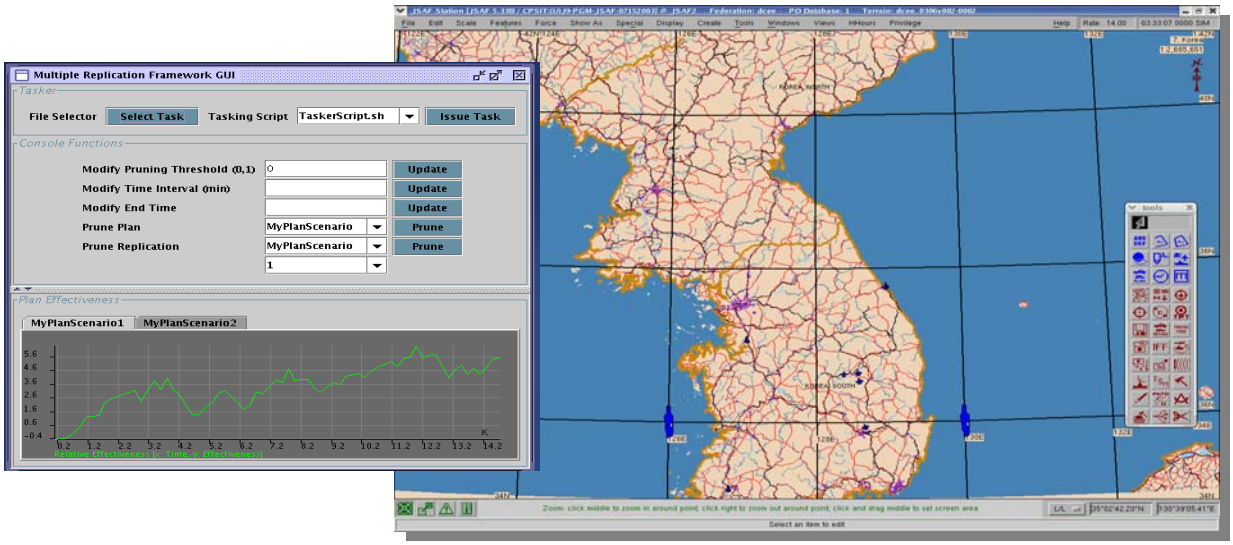


Figure 21: GUI kicking off a JSAF Replication

After the simulation time segment is complete, the MRF automatically performs the plan evaluation and real-time picture evaluation, if the real-time data update was received. Figure 22 shows the plan evaluation results of a replication.

Because of the uncertain nature of predicting the outcome of plans, it is important to execute multiple replications of a plan and statistically analyze the results. The MRF calculates the mean and standard deviation of the effectiveness values for each time step in the simulation. These mean values and their standard deviations can be fitted using $\chi 2$ analysis to obtain time-based curves that provide trend analysis. The $\chi 2$ analysis is used to compare both the simulated and observed results with the expected results of the plan.

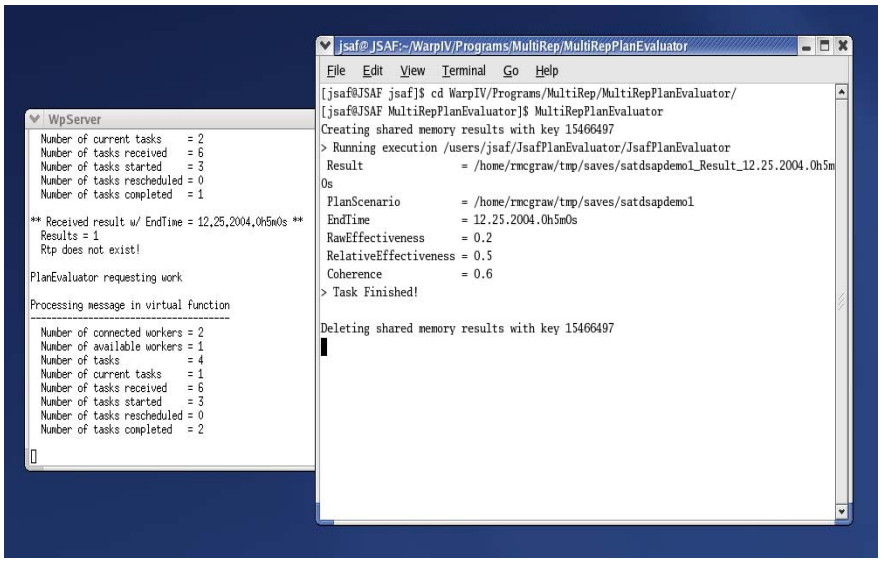


Figure 22: Plan Evaluation Results

6.0 Roadmap for Operating DSAP on the Global Information Grid

One of the overarching goals of this effort is to extend the DSAP Infrastructure to support Network Centric Operations and Warfare (NCOW). To address this goal, the DSAP Infrastructure will be enhanced to work with Network Centric Enterprise Services (NCES) in a Service Oriented Architecture (SOA) in order to support applications utilizing the Global Information Grid (GIG). A starting point for ensuring that DSAP supports the GIG will be to ensure that elements of the infrastructure provide a publish/subscribe capability. This publish/subscribe capability enables processing nodes to (1) publish data or information, (2) subscribe to published data or information, and (3) support query on demand operations for posted data. The operation of the DSAP infrastructure in this context is described as follows and is depicted in Figure 23.

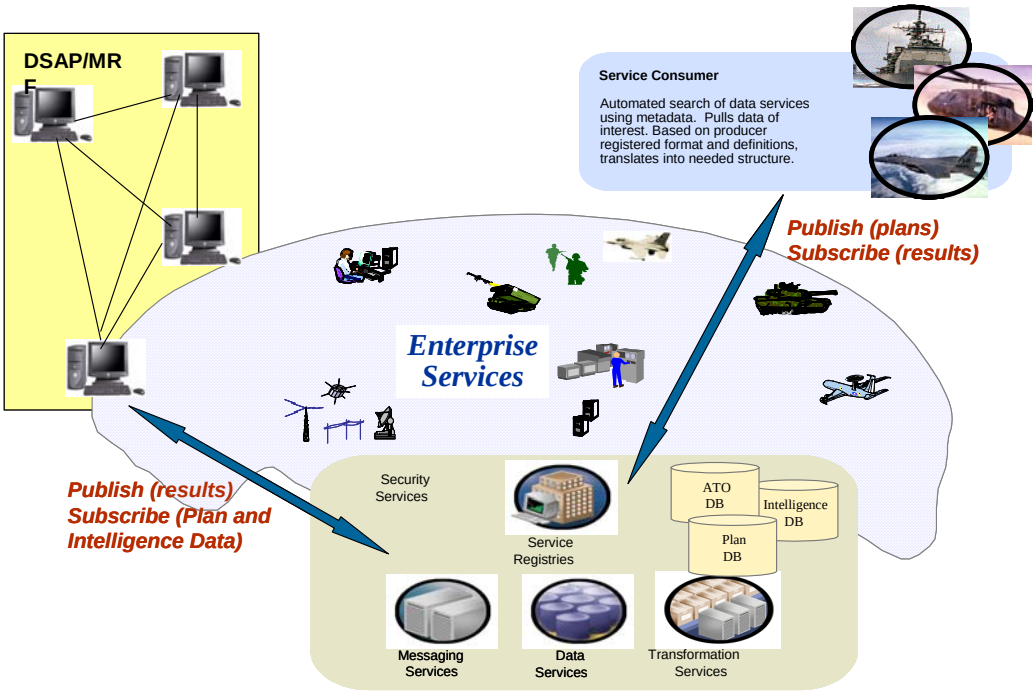


Figure 23: Enhancing DSAP For Use On A Service-Oriented

The underlying communications infrastructure of the DSAP framework is currently being augmented with publish/subscribe functionality. While providing these underlying mechanisms for net-centric operations are needed, much work must be performed to define the data and services that the DSAP Infrastructure will provide to the GIG community.

For example, in their paper on M&S in the GIG environment presented at the IITSEC Conference in 2004, Numrich, Hieb, and Tolk present a view for supporting NCOW in a GIG environment. Among the elements discussed is a value chain based around the concepts of Data Quality, Information Quality, Knowledge Quality, and Awareness Quality. The DSAP Infrastructure can be viewed as a tool for supporting NCOW because it provides measures to support Data Quality (through connections with databases), Information Quality (through connections with real-time feeds), Knowledge Quality (through modeling and simulation using predicted simulation, and Awareness Quality (through the use of its real-time simulation element

as a state estimator). While the DSAP Infrastructure matches up as a support tool for the GIG, support for a web services paradigm is needed. This support is discussed below.

6.1 Web Services

Web services are discrete web-based applications that interact dynamically with other web services. According to Numrich, Hieb, and Tolk four basic definitions are provided and must be met to ensure that a technology or tool can play in a web-service environment. These definitions include:

- Structuring and describing the information to be exchanged
- Specifying the web service
- Accessing and communicating with the web service
- Registering and locating web services

These are described in more detail below.

6.1.1 Structuring and Describing the Information

The first step in defining the necessary data structure and services to make DSAP "GIG-ready" involves structuring and describing the information that will be passed between elements of the DSAP system. This includes defining the structure and type of data necessary to initialize scenarios, describe plans, identify plan objectives and plan priorities, encapsulate plan and replication results, provide plan evaluation information, and extract real-time information between the different services provided by the DSAP Infrastructure. A major emphasis in this area is utilizing a format that can (1) be understood by Commanders and their staff at Air Operations Centers, and (2) interact with tools utilized by Commanders and their staff at Air Operations Centers. The primary mechanism that will be used for describing information in DSAP is the Extensible Markup Language (XML). XML provides the description of data to be exchanged as well its storage and transmission formats. A key facet of XML is that it allows for the definition of schemas that can be used to define supported data types, content, and structure.

With this in mind, several languages and data models are being examined for use in supporting DSAP in a GIG environment. These include: Military Scenario Description Language (MSDL), Battle Management Language (BML), and the Command and Control Information Exchange Data Model (C2IEDM). Each of these models/languages is described below, while outlining their potential use in the DSAP Infrastructure.

6.1.1.1 Military Scenario Description Language (MSDL)

Military Scenario Description Language (MSDL) is a language used to initialize and load scenarios in a simulation environment through the use of an XML based data interchange format. This format enables Command and Control (C2) planning applications to interchange the military portions of scenarios with simulations and other applications. MSDL targets the initialization of simulations and C2 systems with initial state and planned actions.

When implementation in the DSAP Infrastructure is considered, MSDL can be used to define the initial simulation scenarios and initial plans and alternatives that the simulation elements will execute. MSDL can be used to define these scenarios from real-time C4I intelligence data, or to pull representative scenarios from a scenario databases. Use of MSDL will enable DSAP users to

utilize any MSDL-based scenario, or provide scenario information in a standard format to other users on the GIG.

6.1.1.2 Battle Management Language (BML)

The Battle Management Language is a vocabulary or lexicon used by simulation users and developers to specify how to plan and automate military functions in support of Battle Management activities. BML provides a data format for describing military behavior that is derived from military doctrine. The resulting description is a standard that can be passed from human to machine, or machine to machine. BML can be used to (1) describe Command and Control forces and equipment conducting military operations, and (2) provide for situational awareness and a shared common operational picture.

Currently, the DSAP Infrastructure sends plan information, along with objectives and priorities throughout the infrastructure using messages formatted as comma-delimited spreadsheets. These messages are very difficult to decipher and do not adhere to any standard lexicon, making their use by Commanders and their staff at Air Operations Centers very difficult. This effort can utilize BML in support of the DSAP Infrastructure to address these deficiencies by specifying the plans (COAs) and alternatives that will be executed by both the predictive simulation component and the simulation-based state estimator. BML can also be used to specify plan objectives and priorities. These descriptions are derived from military doctrine and will have much greater use when supporting activities at Air Operations Centers.

6.1.1.3 Command and Control Information Exchange Data Model (C2IEDM)

The Command and Control Information Exchange Data Model (C2IEDM) is an information exchange and data management model developed by NATO to specify the structure of information passed between Command and Control Information Systems (C2IS). The C2IEDM preserves the meaning and relationships of information to be exchanged between C2IS at the Conceptual, Logical, and Physical levels.

The DSAP Infrastructure can use the C2IEDM in conjunction with MSDL or BML to specify the underlying relationships that define its entities and actions for its simulation scenarios. The C2IEDM can be used for data interchange at both initialization and in extracting and posting intermediate and final results.

6.1.2 Specifying the Web Services

The second element defined for DSAP will involve specifying and describing the types of web services present in DSAP. The types of web services can be derived directly from elements of the DSAP framework and represent functions such as Worker/Real-time Simulation, Worker/Predictive Simulation, Worker/Evaluator, Worker/Real-time Picture Evaluator, Tasker/Console, Tasker/TBMCS or Real-time Picture. Each service will be described as the type of operation it will perform and the type of data it will handle and return.

6.1.3 Accessing and Communicating with the Web Service

The third element to be defined for the DSAP Infrastructure will be defining the means to access and communicate with its resulting web services. SOAP (Simple Object Access Protocol) is the standard used by the community to define simple one-way mappings for requesting and sending

information. A goal of this effort will be to extrapolate on the current sequence diagrams to define the access and communication sequence required by "web-service" DSAP.

6.1.4 Registration of Web Services

The final element for providing a "web-service" DSAP will be posting and registering the services provided by each service and describing the message size and structure required to satisfy that web service. For each of the services described in 6.1.2, descriptions of the services must be posted and registered.

6.2 Advantages To Using DSAP In A SOA

The preceding subsection has provided a high level overview for enhancing DSAP to play in a SOA. There are many advantages to providing this capability including (1) better support for real-time picture information, (2) access to additional COAs and plans, (3) better proliferation of results to all corners of the GIG, (4) integration with other simulation elements, and (5) improved support for training or live exercises. Each of these is discussed briefly below.

6.2.1 Support for Real-time Picture Calibration

Currently the DSAP Infrastructure calibrates with the real-time picture by connecting to TBMCS (which can be provided through GCCS). The connection is made through JDBC at the MR_TBMCS_Tasker. Currently, real-time picture updates are provided through three methods (1) polling TBMCS databases to see when updates are made, (2) pulling information from the TBMCS databases at some pre-determined "update interval", and (3) subscribing to TBMCS updates. As services currently provided via TBMCS evolve and are subsequently provided as web services in a SOA architecture, TBMCS will solely rely on the DSAP publish/subscribe mechanism to retrieve timely updates that it has subscribed to. This capability can be used to select and filter on specific fields from operations and intelligence databases. The prototype allows information to be pulled from the AODB to provide additional Air Tasking Orders (ATOs), and from the MIDB in order to obtain updated Red Force target information. A goal of this effort, however, will be to extend the real-time update capability of the DSAP framework to handle real-time data feeds by defining and handling the message structure provided by these services. This capability will allow us to utilize more universal information and take advantage of information present across the GIG.

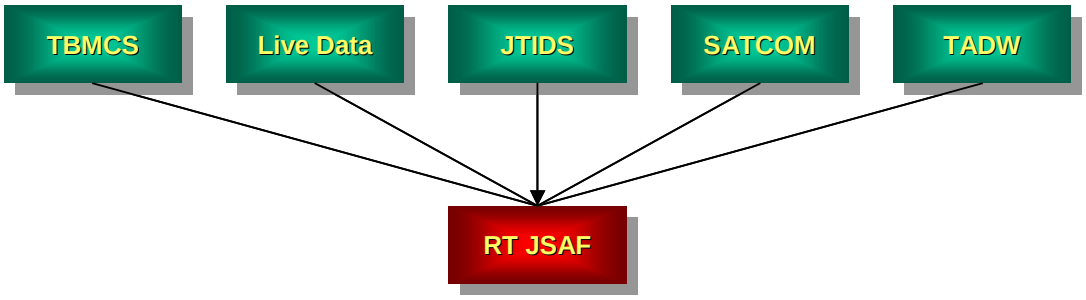


Figure 24: Using Live Data Feeds To Enhance State Estimation

An example of this can be illustrated in Figure 24. The DSAP prototype pulls information from the AODB and MIDB. The information provided from these databases, however, only serves to calibrate the real-time state estimation simulation in terms of Red Force targets and Blue Force ATOs. A key component in evaluating the operational picture includes Blue Force position and

targeting information, which may dynamically change as missions unfold to address Time Critical Targets, Time Sensitive Targets, and other Targets of Opportunity. To understand the effects of dynamic re-targeting and re-tasking of assets and resources on the battlefield, the DSAP prototype must calibrate itself with available data feeds such as JTIDS, SATCOM links, Link-16, etc. A future goal of this effort will be to extend the DSAP's capability to take advantage of both these and similar resources available across the GIG.

6.2.2 Support for Storing and Assessing Alternate COAs

A second key area where DSAP must consider the "artifacts" of playing in a GIG environment involves support for storing and assessing alternate COAs. Currently the DSAP Infrastructure provides a mechanism at its MR_Manager for storing alternate COAs. As the analysis and re-planning process evolves, ineffective COAs may be replaced by plans stored in the MR_Manager's database. To extend DSAP functionality to a SOA, enhancements will be made to both define and support a message structure that will allow COA databases at both local and remote locations to retrieve and contribute the latest plans. This feature will support heterogeneous platforms and better enable system scalability.

6.2.3 Proliferation of Results

DSAP framework must also consider information it may wish to "push" or publish for users of the GIG that may utilize those results for planning and analysis purposes. Currently, results regarding the effectiveness measurements of each plan evaluated by the DSAP infrastructure are stored in a database residing at the MR_Manager and set back to the user/analysis at both the MR_Tasker terminal and console. In order to provide this information to other users, we can define the message structure that will support a service that provides the results of the simulation and evaluation executions through the DSAP infrastructure. This type of intermediate knowledge may have uses beyond the current evaluation process.

6.2.4 Integration With Other Simulations for Analysis

The prototype detailed in this paper uses JSAF for analyzing its COAs. The DSAP infrastructure, however, is not limited to JSAF for analysis. The MRF provides a very non-intrusive mechanism that allows its workers to "fire-off" a JSAF replication. This mechanism can be applied to other simulations such as the Force Structure Simulation (FSS) or the Joint Wargaming System (JWARS) in order to provide COA assessment. A goal to a "web-service" DSAP capability may be to make all simulations web services that can be interchangeably used to populate the DSAP process. Steps in providing this capability would involve defining the structure for scenario initialization data as well as results data.

7.0 DSAP Usage and Future Work

As stated earlier, the DSAP Infrastructure can be used to support analysis, training, and operations. In terms of Analysis, the DSAP environment can be used for Course of Action Analysis and rapid mission rehearsal. The MRF's Prediction capability can be used to analyze COAs and potential alternatives by simulating those plans faster-than-real-time. Raw effectiveness calculations can then be used to evaluate and rank each plan or replications of each plan to determine its effectiveness at achieving its goals and objectives within its operating context.

The DSAP Infrastructure can be used to support Training of Commanders and their staff through human-in-the-loop simulation. In this context, the Commander can be trained using the real-time simulation in the absence of the calibrated results. This training process utilizes the predictive analysis of plans (using faster-than-real-time simulation) to provide the Commander and his staff with feedback on the potential effectiveness of a plan and possible alternatives. The MRF can allow the Commander to select a new plan, analyze a new alternative, or prune failed alternatives based on feedback provided to him. This training process can be extended to live environments by incorporating the real-time C4I aspects of the framework.

The DSAP Infrastructure can be used to support Operations in a number of ways. First, the DSAP Infrastructure can be used to support the planning process. The Prediction capability can be used to rapidly rehearse plans faster-than-real-time via simulation and provide measures of plan effectiveness as input to plan generation tools. In this context, the DSAP Framework can take the plan generation process from simply an optimization process that "covers" plan objectives to a process that incorporates mission rehearsal for a more accurate optimization process.

Second, the full DSAP capability is geared toward use at Air Operations Centers in an operational environment. In this instance, the DSAP Infrastructure can take mission plans and potential alternatives generated from Air Tasking Orders (ATOs) and use the Prediction capability to (1) predict their effectiveness in the future while calibrating with the current real-time picture, and (2) support Dynamic Situation Assessment via state estimation. In this context, the DSAP Infrastructure can be used to evaluate and replace plans as the operational picture evolves.

7.1 Future Work

To continue to build on successes of our DSAP work, RAM Laboratories is proposing to continue to build on MRF functionality and to integrate and install a working prototype of the DSAP Infrastructure in the laboratory at AFRL's Rome Research Site. Specifically we propose develop, implement, and integrate the real-time operationally focused simulation component of DSAP in an operational setting to provide estimation of real-world state. In addition, we propose to augment our existing predictive analysis capabilities and Graphical User Interfaces to facilitate DSAP's use in predicting the outcomes of plans and alternatives through "what-if" analysis.

8.0 Bibliography

John R. Surdu. Connecting Simulation to the Mission Operational Environment. Ph.D. Thesis. Texas A&M University. 2000

Alex F. Sisti. “Dynamic Situation Assessment and Prediction (DSAP)” Proceedings of SPIE, Enabling Technologies for Simulation Science VII Vol.5091. 2003.

Dr. Paul Phister, Dr. Timothy Busch, and Igor Plonisch. “Joint Synthetic Battlespace: Cornerstone for Predictive Battlespace Awareness.”

Reaper Jerome, Trevisani Dawn, and Alex Sisti. “Real-Time Decision Support System (RTDSS)” Proceedings of the Western MultiConference. January, 2003.

McGraw Robert, Lammers Craig, and Steinman Jeff, 2004. “Software Framework in Support of Dynamic Situation Assessment and Predictive Capabilities for JSB-RD”. In proceedings of the SPIE - Enabling Technologies for Simulation Science VIII Conference.

McGraw Robert, Lammers Craig, and Trevisani Dawn, 2004. “Dynamic Situation Assessment and Predictive Capabilities in Support of Operations”. In proceedings of the Fall Simulation Interoperability Workshop, Orlando, FL. 2004.

McGraw Robert, Lammers Craig, Steinman Jeff, and Trevisani Dawn, 2005. “A DSAP Infrastructure for the Global Information Grid's Modeling and Simulation Community of Interest”. In proceedings of the *Spring Simulation Interoperability Workshop,* San Diego, CA. 2005.

Lammers Craig, McGraw Robert, and Trevisani Dawn, 2005. “Applying a Multireplication Framework to Support Dynamic Situation Assessment and Predictive Capabilities”. In proceedings of the SPIE - Enabling Technologies for Simulation Science IX, Orlando, FL. 2005.

Effects Based Operations. Available: http://www.afrlhorizons.com/Briefs/June01/IF00015.html

Theater Battle Management Core Systems (TBMCS). Available http://jitc.fhu.disa.mil/tbmcs/tbmcs.htm.

Available http://www.mstp.quantico.usmc.mil/modssm2/InfoPapers/INFOPAPER%20JSAF.htm

Numrich, S.K., Hieb, M., and Tolk, A. “M&S in the GIG environment: An Expanded View of Distributing Simulation” Presented at the Interservice Industry Training Simulation Education Conference. Orlando, FL. 2004.

http://e-mapsys.com/C2IEDM-MIP_Overview_20Nov2003.pdf

]Steinman Jeff, 2002. “The Standard Simulation Architecture.” In proceedings of the 2002 SCS Summer Computer Simulation Conference.

Douglas Schmidt. ACE+TAO. Available http://www.cs.wustl.edu/~schmidt/.

Bailey Chris, McGraw Robert, Steinman Jeff, and Wong Jennifer, 2001. "SPEEDES: A Brief Overview" In Proceedings of SPIE, Enabling Technologies for Simulation Science V, Pages 190-201.

RAM Object Request Broker Programming Guide, Version 1.2, DRAFT.

9.0 Acronyms

ACE	Adaptive Communication Environment
AODB	Air Operations Data Base
AFRL	Air Force Research Laboratory
ATO	Air Tasking Order
BML	Battle Management Language
C2IEDM	Command and Control Information Exchange Data Model
C2IS	Command and Control Information Systems
C4I	Command, Control, Communications, Computers, and Intelligence
CCSE	Common Component Simulation Engine
COA	Course Of Action
CORBA	Common Object Request Broker Architecture
DSAP	Dynamic Situation Assessment and Predictive
FSS	Force Structure Simulation
FTRT	Faster Than Real Time
GCCS	Global Command and Control System
GIG	Global Information Grid
GUI	Graphical User Interface
IFSB	Information Systems Branch
IITSEC	Interservice Industry Training Simulation Education Conference
JDBC	Java Data Base Connectivity
JDK	Java Developer's Kit
JSAF	Joint Semi-Automated Forces
JTIDS	Joint Tactical Information Delivery System
JWARS	Joint WarGaming System
MIDB	Modernized Integrated Data Base
MRF	Multiple Replication Framework
MSDL	Military Scenario Description Language
NATO	North Atlantic Treaty Organization
NCES	Network Center Enterprise Services
NCOW	Network Centric Operations and Warfare

ODBC	Open Data Base Connectivity
ORB	Object Request Broker
OS	Operating System
POC	Point of Contact
RT	Real Time
RTP	Real Time Picture
SATCOM	Satellite Communications
SBIR	Small Business Innovative Research
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SPEEDES	Synchronous Parallel Environment for Emulation and Discrete Event Simulation
TAO	The ACE ORB
TBMCS	Theater Battle Management Core System
XML	Extensible Markup Language